«ИНФОРКОМ»

ZX-РЕВЮ
1992

Выпуск 01-12

VERSION 1.00

PDF version by NUK, km, Василий Кормилицын, Cybrex, Александр Кульнев.

Feedback: nuk@mail.ru

ZX-РЕВЮ № 1-2'1992 "The Second Encounter" Completed
ZX-РЕВЮ № 3-4'1992 "Byte Eater" Completed
ZX-РЕВЮ № 5-6'1992 "Top Secret" Completed
ZX-РЕВЮ № 7-8'1992 "Das Dunkle Rad" Completed
ZX-РЕВЮ № 9-10'1992 "Krieg Der Sterne" Completed
ZX-РЕВЮ № 11-12'1992 "Empire Strikes Back" Completed

Спектрум в школе	2[1]	Completed	by Cybrex
Beta Basic	3	Completed	by Cybrex
Защита программ	9	Completed	by km
40 лучших процедур	17	Completed	by km
40 лучших процедур	21	Completed	by Cybrex
Мастерфайл-09 полная русификация	29	Completed	by Cybrex
Stalingrad	33	Completed	by Александр Кульнев
Flight Simulation	35	Completed	by Александр Кульнев
Star Raiders 2	36	Completed	by Александр Кульнев
The Train	37	Completed	by Александр Кульнев
Death Wish 3	38	Completed	by Александр Кульнев
Forum	39	Completed	by Василий Кормилицын
Спектрум в школе	45	Completed	by Cybrex
POKES	46	Completed	by Cybrex
Beta Basic	47	Completed	by Cybrex
Защита программ	53	Completed	by Cybrex
40 лучших процедур	61	Completed	by Cybrex
Мастерфайл-09 полная русификация	71	Completed	by Cybrex
Professional Tennis	77	Completed	by km
Snooker	77	Completed	by km
Quazatron	79	Completed	by km
Captain Fizz	82	Completed	by km
Forum	84	Completed	by km
Спектрум в школе	89	Completed	by Cybrex
BetaBasic.	91	Completed	by Cybrex
Защита программ.Глава 4.	97	Completed	by km
Защита программ.Том 2.Глава 1-2	100	Completed	by Василий Кормилицын
40 лучших процедур	105	Completed	by Cybrex
Каналы и потоки (Возвращаясь к напечатанному)	111	Completed	by Василий Кормилицын
Профессиональный подход	113	Completed	by km
Forum	117	Completed	by NUK
Forum	121	Completed	by Василий Кормилицын
Academy	125	Completed	by km
Sherlock	129	Completed	by km
Наш конкурс	131	Completed	by km
ZX-РЕВЮ № 7-8,9-10,11-12'1992		Completed	by NUK

[1] Номера страниц в оригинальном издании

Содержание



107241, Москва, Б-241, а/я 37, "ИНФОРКОМ"

Дорогие друзья!

Мы встречаем второй год существования "ZX-РЕВЮ". Позвольте поблагодарить наших верных читателей прошлого года, подписавшихся и на этот. Благодарим также и тех, кто в это трудное время поверил в нас, поверил в то, что мы будем жить и работать и впервые подписался на наше издание.

Мы, конечно, сохраним тот дух и стиль, которые были достигнуты в прошлом году, но кое-какие незначительные изменения все-таки внесем. Во-первых, уже в этом номере Вы увидите больше крупных учебных материалов, этим мы несколько парируем ту раздробленность, которая была в прошлом году, но в прошлом году мы готовили материал с "колес", а к этому году подошли с определенным портфелем и теперь можем давать более крупные вещи. В связи с этим будет как бы меньше число разделов в одном номере, но каждый из них будет крупнее. По всей видимости, это вызовет также необходимость полностью перейти на практику выпуска двойных номеров. Это, кстати, несколько поможет нам чуть-чуть снизить потери от резко возросших почтовых тарифов.

Поскольку в проекте намечены крупные материалы, мы будем их давать блоками, содержащими четное количество страниц. Это позволит Вам при желании без проблем расшивать выпуски и комплектовать свои подшивки по отдельным работам. Об этом просили многие читатели уже давно, но только сейчас появилась такая физическая возможность.

В этом году несколько больший упор будет сделан на основы программирования в машинных кодах и на Ассемблере.

Скорее всего, несколько уменьшится объем материалов технического характера, аппаратурных вопросов. Вы понимаете, что это связано, прежде всего, с тем, что мы не занимаемся аппаратным обеспечением и пользовались только тем, что нам давали партнеры.

А в общем "ZX-РЕВЮ" остаётся тем же, каким и было.

Мы по-прежнему, как и год назад, гарантируем всем читателям, подписавшимся на "РЕВЮ" возврат стоимости подписки, если кто-либо разочаруется в содержании этого номера и вернет нам его наложенным платежом, оценив в стоимость своих затрат и уведомив нас об этом письмом.

Многих волнует вопрос, как мы выйдем из того положения, в котором оказалась печать в масштабах всей страны, Вы ведь знаете, что вздорожавшая в прошлом году в 10 раз бумага сейчас поднялась еще в 5 раз. Невыносимыми стали почтовые тарифы. Достаточно сказать, что одна бандероль в Прибалтику теперь может стоить до 30 рублей. Люди волнуются, предлагают помощь. Многие пишут, что если нам будет совсем невмоготу, то они лучше доплатят, чем мы закроем наше дело. Большое спасибо, уважаемые читатели за вашу заботу, но пока мы держимся.

Наш принцип такой: "Когда дует сильный ветер перемен, то надо строить не забор, а мельницу".

Не успев начать подписной год, центральная пресса громко заголосила о невозможности выполнить взятые перед подписчиками обязательства и добилась дотаций от правительства, хотя голосить не перестала. Нам такой подход кажется не очень

красивым. Мы же работаем над другими проектами и сейчас для нас пока нет веских оснований предполагать, что нам придется просить Вас делать какие- либо доплаты, если мы нормально развернем маркетинг программного обеспечения для IBM. Хотя, конечно, в этом вопросе мы очень и очень рассчитываем на Вашу поддержку.

В №11-12 за прошлый год мы подробно расписали, как мы будем строить нашу с вами деятельность, предполагая начислять тем из Вас, кто помогает продвижению наших продуктов по 8% от объема продаж, проведённых по результатам Вашего исследования местного рынка.

Так же мы объявили о программном комплексе "АНГРАМ" (полный курс английского языка в 22-х программах) и предложили потенциальным дистрибуторам приобретать для представительства так называемый "ЗЕЛЕНЫЙ ПАКЕТ", в который входит образец программного средства, рекламно-информационные материалы, бланки договоров, инструкции и т.п.

Сейчас, когда пишутся эти строки, процесс уже пошел и по своим темпам превосходит наши самые смелые ожидания. Работа с дистрибуторами пошла. Если темп будет нарастать так, как сегодня, мы к осени этого года введем внутреннее специализированное издание, распространяющееся бесплатно "среди своих", содержащее анализ работы, обмен передовым опытом и описания наиболее сложных игровых программ для IBM-совместимых машин в качестве развлекательной части. Предполагаем, что записывать понравившиеся им программы наши дистрибуторы будут у нас бесплатно, а коллекция у нас - немалая. Одним словом, определяется наша маркетинговая политика на многие ближайшие годы.

Сегодня мы представляем на последней странице "ZX-РЕВЮ" не обучающую программу, а информационно-поисковую систему "РЕГИСТРАТУРА". Она выбрана для представления, поскольку обладает наиболее универсальным характером возможного применения. Наверное, очень трудно представить организацию, в которой был бы компьютер, чтобы она не нуждалась в подобной системе (разве что аналогичная уже есть). Система проходит опытную эксплуатацию уже более года в разных организациях в качестве системы учета кадров, системы ведения штатного расписания, системы учета клиентов в малом предприятии, системы учета обратившихся в медицинское учреждение, системы учета движения документов в делопроизводстве райисполкома, системы учета нуждающихся ветеранов войны и труда, системы учета учащихся на факультетах высшего учебного заведения. Мы внедрили ее даже на одном машиностроительном предприятии для учета заготовок. Это то, что уже есть на практике, а теоретически можно учитывать все и всюду.

Имеющийся опыт эксплуатации позволяет однозначно сказать, что среди систем с аналогичными возможностями, нашу выделяет необычайная простота в освоении и эксплуатации, она сделана для тех, кто впервые в своей жизни увидел компьютер. В этом мы видим свое главное достижение. Имея опыт преподавательской работы и методологию подготовки обучающих программ, мы и деловые системы делаем с расчетом на то, чтобы осваивать их можно было без головной боли и без необходимости изучать объемную сопроводительную документацию.

Далее, в последующих выпусках ZX-РЕВЮ мы будем чередовать представление обучающих и деловых систем. Ждем Вашего участия в дистрибуторской сети.

В заключение мы должны извиниться за возможную нерегулярность выпусков. Хоть мы и с оптимизмом смотрим в будущее, нам все-таки приходится, где возможно экономить и, к сожалению, мы вынуждены печатать тираж не тогда когда надо, а тогда, когда удается купить бумагу по более-менее сносным ценам, т.е. мы, конечно, не упускаем возможности уменьшить свои издержки. В то же время, нерегулярность не означает хроническое отставание, иногда мы, наверное, будем и забегать вперед.

Спектрум в школе

Сегодня раздел для начинающих представляют две БЕЙСИК-программы с комментариями, подготовленные Черкасским В. А. (г. Борисов).

Программы не сложны, но каждая из них содержит ошибки, попробуйте их отыскать.

1. "LISTNAME"

Эта программа записывается в начале кассеты и при работе выводит названия программ и их местоположение по счетчику.

```
10 DIM A$(50,3): DIM B$(50,10):DIM C$(50,4)
20 BRIGHT 1:  CLS: PRINT #0;INK 1: PAPER 6; AT 0,1;
     "BORISOV  $ ALCOSOFT $ 1991 ": PRINT AT 8,8;
     "1. Edit check"; AT  9,8;
     "2. Looking"; AT 10,8;
     "3. Save to the tape"
30 IF INKEY$="3" THEN GO TO 140
40 IF INKEY$="1" THEN GO TO 70
50 IF INKEY$="2" THEN GO TO 100
60 GO TO 30
70 CLS: PRINT " '0'-END"
80 INPUT "Number=";i:
     PRINT "      ";i;TAB(0):
     INPUT "Count="; A$(i)'
     "Name=";B$(i)'
     "DATA=";C$(i): GO SUB 150
90 GO TO 80
100 CLS: PRINT "Count      Name      Data": FOR  i=1 TO 50
110 BEEP .07,35: IF B$(i)="  " THEN PAUSE 0: GO TO 20
120 GO SUB 150
130 NEXT i
140 CLS: PRINT AT 10,10; "Save programs":SAVE  "LISTNAME" LINE 100:
     CLS: PRINT AT 10,10; "VERIFY programs": VERIFY  "LISTNAME":
     GO TO 20
150 PRINT A$(i);"_____"; B$(i); "_____";C$(i): RETURN
```

Пояснения к программе:

10 задаем строковые массивы для переменных:

- A$ - показания счетчика;
- B$ - названия программ;
- C$ - год выпуска программы.

20 Выводим меню.

30-50 Выбор режима работы.

В зависимости от нажатой клавиши управление передается на разные строки программы.

60 Переход на строку 30 для организации ожидания нажатия клавиши.

70 Стирает экран, выводит условие выхода из режима.

80 Ожидается ввод порядкового номера, вывод его на экран и последовательно ввод показания счетчика, названия программы, года выпуска. Заканчивается обращением к подпрограмме вывода переменных на экран.

90 Организуется ввод каталога.

100 Инициализация экрана, вывод заглавия, начало цикла вывода данных.

110 Подача звукового сигнала, проверка на "пустое название", если "да", то пауза до нажатия любой клавиши и возврат в меню.

120 Обращение к подпрограмме вывода переменных.

130 Конец цикла FOR…NEXT. Пауза до нажатия клавиши и возврат в меню.

140 Очистка экрана, запись на ленту с последующим автостартом программы с 100 строки, очистка экрана, проверка записанной программы, возврат в меню.

150 Подпрограмма вывода переменных.

Ошибки:

Нужно добавить строку

```
145 IF i=0 THEN GO TO 100
```

и в строке 80 добавить после команды

```
INPUT "Number=";i : GO SUB 145
```

Это даст нормальный выход при программировании каталога и выведет название, которое пользователь будет редактировать. Кроме того, в 110 строке сравнение нужно проводить не с " ", а с "   ", т.к. переменная B$(i) имеет фиксированную длину в 10 символов - это имя программы.

2. "REMONT"

Эта программа может служить пособием по ремонту различной техники. Причем, когда у машины варианты кончатся, то она спросит у пользователя, что неисправно, и в следующий раз она задаст и этот вопрос. Таким образом, постепенно количество вариантов увеличится.

```
5 GO TO 20
10 LOAD  ""CODE
20 DIM A$(500,32)
30 LET  i=1: LET A$(i)=" напряжение батареек"
40 CLS:  POKE 23607,249: PRINT "Я думаю, стоит проверить ";A$(i): GO SUB 140
50 IF UP=0 THEN PRINT FLASH  1';" Нужно ремонтировать!":PAUSE 0: GO TO  90
60 LET  i=i+1:IF A$(i)="  " THEN GO TO  80
70 GO TO 40
80 INPUT "Не хватает вариантов "' " как вы думаете, что еще нужно проверить ? " 'A$(i)
90 POKE 23607,60: CLS: PRINT AT 21,8; "Press any Key !": PRINT #0:
      INK 1; PAPER 6; AT 0,1;" BORISOV $ ALCOSOFT $ 1991":
      POKE 23607,249:PRINT AT 5,3;
      "1. - Работа с программой"; AT 5,3;
      "2. - Запись на магнитофон"; AT 7,3;
      "3. - Проверка записи"
100 IF INKEY$="1" THEN GO TO 30
110 IF INKEY$="2" THEN SAVE "REMONT" LINE 10
120 IF INKEY$="3" THEN VERIFY "REMONT"
130 GO TO 100
140 PRINT ," В норме?"'" Да/Нет?"
150 IF PEEK 23560=100 THEN LET UP=1:RETURN
160 IF PEEK 23560=110 THEN LET UP=0:RETURN
170 GO TO 150
```

Описание

5 Служит для того, чтобы после команды RUN программа нормально стартовала, а не производила загрузку по строке 10.

10 Загружает коды знакогенератора русского алфавита.

20 Объявляет строковый массив A$.

30 Объявляем переменную i и элемент массива A$(i).

40 Очищаем экран, переключаемся на русский шрифт, выводим вопрос и обращаемся к подпрограмме управления.

50 Если неисправность определена правильно, то выдаем сообщение.

60 Прибавляем 1 к i. Берем переменную A$(i) и проверяем ее на пустоту.

70 Организовываем переход для следующего вопроса.

80 Ждем вопроса, который вводит пользователь.

90 Вывод меню.

100 Опрос клавиши "1". Если нажата, то работа с программой.

110 То же клавиши "2". Если нажата - выгрузка программы на ленту.

120 То же клавиши "3". Если нажата - проверка программы.

130 Организовывает ожидание INKEY$.

140 Начало подпрограммы определения ответа.

150 Опрашивается клавиша "D".

160 То же - клавиша "H".

170 Организация ожидания INKEY$. Знакогенератор находится в памяти по адресам 64000... 64768.

Ошибки.

1. В строку 150 надо дописать POKE 23560,0 для того ,чтобы программа не проскакивала INKEY$.

2. В строке 60 сравнение должно быть с пустой строкой длиной 32 символа.

BETA BASIC

БЕТА-БЕЙСИК 3.0 - дальнейшее развитие серии, начатое программами БЕТА - БЕЙСИК 1.0 и 1.8. Как и последние, он полностью совместим со стандартным, зашитым в ПЗУ вашего компьютера БЕЙСИКОМ, но обладает значительно большими возможностями, определенной гибкостью и имеет значительное число новых операторов и функций.

БЕТА-БЕЙСИК откроет перед вами гигантские перспективы. Во-первых, с его помощью Вы можете писать большие программы. Если, Вы не пробовали на обычном БЕЙСИКе писать программы более 10 - 15 К, то знайте, что "проклятьем" обычного БЕЙСИКа является отсутствие структурного программирования, наступает момент, когда программа не укладывается, как цельное произведение в вашем мозгу и тогда любые изменения, производимые в одном месте программы, влекут за собой помехи, появляющиеся в другом месте, как бы ни был прост БЕЙСИК в освоении, но в отладке большие программы, написанные на БЕЙСИКе, могут выходить за все разумные рамки по трудоемкости.

Второе преимущество БЕТА-БЕЙСИКА - отличные средства редактирования. Этот момент, надо сказать, ахиллесова пята стандартного БЕЙСИКа "Спектрума". Конечно, строчный редактор со сложной системой вызова строки на обработку - это не дело. Только на этом программист теряет половину своего драгоценного времени. В то же время, полноэкранные редакторы, какие мы виден в БЕЙСИКе на хорошо продвинутых машинах типа MSX или, скажем, IBM (GW_BASIC) - это тоже не подарок. Перемешанные на одном экране редактируемые и нередактируемые строки приводят к многочисленным ошибкам, да и занимают такие редакторы много места.

Тот подход, который развит в БЕТА-БЕЙСИК 3.0 наверное наиболее практичный и эргономичный.

Третье преимущество БЕТА-БЕЙСИКА - в его экономном расходовании памяти. Да, конечно, сам он занимает до 20К. То есть почти половину того, что имеет Ваш компьютер. Но, если Вы, программируя, будете хорошо использовать его возможности, то затратите 4-5 К на основные процедуры, а далее, используя их, сможете в каждый килобайт укладывать то, что на обычном БЕЙСИКе стоило бы вам пяти-семи К. Таким образом, в доступных вам 20 килобайтах, Вы сможете разместить примерно 10К данных плюс 4-5К процедур, плюс программу, эквивалентную обычным 40К стандартного БЕЙСИКа. Конечно, это цифры условные, но условность их в том, что это далеко не предел. Поработав несколько недель, Вы сможете еще сильнее повысить емкость программ, это просто данные нашего собственного практического опыта в "Инфоркоме".

Данный документ является нашим авторизованным переводом фирменной инструкции. Авторизованность не означает, что мы что-то сократили. Мы чуть более подробно прокомментировали наиболее узкие места и заменили (там, где это можно) те примеры, которые были привязаны к микродрайву на более близкие для нашего читателя.

Пусть Вас не смущает объем документа, нет никакой необходимости читать все от начала и до конца, вполне достаточно, если Вы прочтете введение, раздел по редактированию программ и, в дальнейшем, будете обращаться к документу только при необходимости использовать тот или иной оператор или ту или иную, ранее неизвестную Вам функцию. Особо же рекомендуем вам обратить внимание на те преимущества, которые предоставляет структурное программирование благодаря использованию процедур.

ГЛАВА 1. ВВЕДЕНИЕ

Загрузка языка с ленты выполняется обычным образом:

```
LOAD "" или LOAD "Beta Basic"
```

По этой команде загружаются три первых БЕЙСИК-строки - строка 0, строка 1 и строка 2. Со строки 2 осуществляется автостарт программы, в результате которого загружается остальная (основная) часть БЕТА-БЕЙСИКа, представляющая из себя блок машинных кодов длиной 18К. Системная переменная RAMTOP понижается до значения

47070, предохраняя от повреждения машинный код, размещаемый в верхней части памяти.

После загрузки программы на экране должно появиться исходное системное сообщение. При этом строки 1 и 2 программы удаляются и остается только нулевая строка.

Внимание! Если Вы написали программу под управлением БЕТА-БЕЙСИКа и желаете передать ее своим знакомым для эксплуатации, то пожалуйста перед выгрузкой Вашей программы и машиннокодового блока внесите следующие изменения в машиннокодовую часть БЕТА-БЕЙСИКа:

```
POKE 64218,0: POKE 64219,0: POKE 64220,0: POKE 64861,201
```

Этим Вы обеспечите то, что Ваша программа будет запускаться, но не сможет быть изменена.

Особенности работы с языком БЕТА-БЕЙСИК 3.0

В строке 0 содержатся новые определения функций БЕТА-БЕЙСИКа 3.0. Эта строка всегда присутствует в программе, хотя и не всегда изображается на экране. Характерным свидетельством того, что БЕТА-БЕЙСИК загружен и готов к работе является изображение указателя текущей строки в инвертированном виде. Несколько увеличена и продолжительность звукового сигнала, подаваемого при нажатии клавиш. Если он вам не нужен, дайте команду POKE 23609,0 для отключения. Теперь можете загружать любую БЕЙСИК-программу (доступного размера) и она будет работать нормально, за исключением двух особенностей:

1. Загрузку программы надо выполнять не командой LOAD, а командой MERGE, иначе погибнет нулевая строка. Конечно, если программа была написана под стандартным БЕЙСИКОМ, ей это все равно, но если под БЕТА-БЕЙСИКОМ, строку 0 уничтожать не надо.

2. Если вам надо, чтобы вместо ключевых слов БЕТА-БЕЙСИКа на экране изображались символы блочной графики, надо дать команду KEYWORDS 0 (см. соответствующую команду).

Еще одной особенностью является то, что несколько изменилось изображение программных строк, получаемое на экране по команде LIST, если длина строки больше ширины экрана. В этом случае в первых четырех позициях печатаются только номера строк.

Несколько уменьшено разрушительное действие команды NEW. Она удаляет из памяти имеющуюся там БЕЙСИК-программу, но оставляет строку 0. Если же Вы хотите удалить из памяти и сам БЕТА-БЕЙСИК, то остается только сделать RESET, если у Вас есть такая кнопка, или выключить питание или переинициализировать машину командой RANDOMIZE USR 0.

Несколько увеличена скорость выполнения программ по сравнению со стандартным БЕЙСИКОМ, особенно длинных.

Циклы FOR-NEXT работают с одной и той же скоростью, независимо от местоположения в программе, что является отличием от стандартного БЕЙСИКа. Наивысшая скорость достигается, когда начало цикла, его длина и конец выражены целыми числами (в том числе и отрицательными) размером до 65535. Такие циклы работают в 5 раз быстрее, по сравнению с циклами, размещенными до 100-й строки в стандартном БЕЙСИКе и в 17 раз быстрее, по сравнению с размещенными до 500-ой строки в стандартном БЕЙСИКе.

Быстрее работают и GO TO и GO SUB, особенно когда строка назначения отстоит достаточно далеко. Быстрее работает и RETURN. Возврат к последней строке выполняется столь же быстро, как и к первой, в отличие от стандартного БЕЙСИКа.

В какой-то степени скорость удалось повысить за счет того, что интерпретатор запоминает адреса, а не номера строк. С другой стороны, это накладывает определенные ограничения на редактирование. Так, если Вы, например, остановите программу во время исполнения цикла, введете внутрь этого цикла дополнительную строку, а потом запустите программу дальше - CONTINUE, то CONTINUE может не сработать, поскольку адрес возврата изменится, хотя номер строки возврата не изменялся.

Работа с более ранними версиями БЕТА - БЕЙСИКа

Если Вы загрузите программу, написанную в более ранней версии БЕТА-БЕЙСИК 1.0

или БЕТА-БЕЙСИК 1.8, то вместе с ней загрузится и нулевая строка, содержащая определения функций языка. Если теперь Вы хотите работать с этой программой под управлением версии 3.0, то Вам надо заменить старую версию нулевой строки на новую.

1. Загрузить БЕТА-БЕЙСИК 3.0.
2. Загрузить программу, выполненную в версии 1.0 или 1.8.
3. Выполнить "MERGE" для БЕТА-БЕЙСИКа 3.0, чтобы ввести копию строки 0.
4. Удалить строки 1 и 2.
5. Выгрузить полученную программу.

Есть два момента несовместимости версии 3.0 с ранними версиями. Во-первых, переменная под именем "line", создававшаяся операторами ON ERROR и TRACE, теперь называется "lino", чтобы исключить путаницу с ключевым словом стандартного БЕЙСИКА LINE.

Во-вторых, в новой версии имена процедур не могут иметь внутренних пробелов.

Преобразовать Вашу программу из версии, написанной на более ранней версии языка, Вы можете воспользовавшись командой ALTER.

Для тех, кто знает версии 1.0 и 1.8, желательно перечитать работу с командами ON, RETURN, ROLL, SCROLL, CLOCK, ON ERROR и TRACE, поскольку они изменены. Несколько изменена и команда GET (число). Значительно расширены возможности процедур.

Загрузка и выгрузка программ, написанных в БЕТА-БЕЙСИКе 3.0.

После того, как Вы написали программу, в которую входят некоторые новые ключевые слова, Вы можете выгрузить ее на ленту обычным порядком. Если теперь вам когда-либо понадобится вновь загрузить ее, сначала загрузите БЕТА-БЕЙСИК 3.0, если он еще не загружен. Теперь загрузите (LOAD) свою программу. Поскольку строка 0 была выгружена вместе с программой, то теперь нет необходимости использовать MERGE.

С другой стороны, Вы можете воспользоваться строками 1 и 2 загрузчика для того, чтобы сохранить на ленте свою программу вместе с машинно-кодовой частью БЕТА - БЕЙСИКа. В этом случае Ваша программа и БЕТА-БЕЙСИК будут загружаться автоматически.

Если Вы загрузите программу, написанную на БЕТА-БЕЙСИКе, в компьютер в то время, как в нем не присутствует машинный код БЕТА-БЕЙСИКа, новые команды появятся в виде одиночных символов, а после команды RUN Вы получите сообщение об ошибке "Nonsense in Basic". В этом случае Вам следует использовать: MERGE "Beta Basic": GO TO 2, чтобы загрузить загрузчик БЕТА БЕЙСИКА, а из него по автостарту посредством GO TO 2 загрузится и машинно-кодовая часть.

ГЛАВА 2. РЕДАКТИРОВАНИЕ

Список используемых ключевых слов: EDIT, KEYWORDS, LIST, FORMAT, CSIZE, JOIN, SPLIT.

БЕТА-БЕЙСИК позволяет делать ввод и редактирование программ намного более простым, чем то, к чему Вы привыкли, работая со стандартным встроенным БЕЙСИКОМ "Спектрума". Если Вы наберете или прильете (MERGE) некую БЕЙСИК программу, то сможете поэкспериментировать с новыми возможностями. Поскольку все возможности стандартного БЕЙСИКа сохранены. Вы вряд ли будете испытывать при этом какие-либо трудности. Ниже мы рассмотрим, как действуют те команды, которые добавляет использование третьей версии БЕТА-БЕЙСИКа при редактировании. Более подробно мы на них остановимся еще раз в основной части руководства.

Курсор текущей строки

Первое, что Вы заметили, загрузив БЕТА-БЕЙСИК, - это курсор текущей строки, изображающийся инверсным цветом на экране. Его можно перемещать с помощью курсорных клавиш вверх и вниз и выполняется это намного быстрее, чем в стандартном БЕЙСИКе, поскольку при этом не листается содержимое всего экрана. (Могут быть редкие случаи, когда положение курсора не вполне соответствует изображению текста программы

на экране. В этом случае просто нажмите ENTER для перестроения экрана).

Команда EDIT <номер строки>

Вы можете вызвать на редактирование даже ту строку, которой в данный момент нет на экране. Теперь нет необходимости подгонять к ней курсор. Достаточно нажать "0" и набрать нужный номер строки, чтобы она появилась в нижней части экрана.

В редактируемой строке Вы теперь можете перемещать курсор не только влево или вправо, но и вверх и вниз, это опять же выполняется курсорными клавишами. При попытке поднять курсор выше верхней строки или опустить ниже нижней, он автоматически встанет в конце строки. Самый быстрый путь добавить операторы в конец Вашей строки - это:

- нажать клавишу "0";
- ввести номер строки;
- нажать "Курсор вверх" - теперь он окажется в конце строки.

Переключение режимов курсора

Теперь Вы можете в нужный момент легко переключить курсор "K" на курсор "L" или "C". Это бывает полезным в тех случаях, когда Вы хотите набрать по буквам имя процедуры или если Вы не желаете вводить ключевые слова как токены, то есть одним нажатием клавиши, а хотите набирать их по буквам, что возможно благодаря команде KEYWORD (см. ниже). Выполняется переход из режима "K" в режим "L/C" нажатием клавиши "пробел".

Возможен и обратный переход из режима "L/C" в режим "K", что выполняется одновременным нажатием клавиш SYMBOL SHIFT и ENTER. Те, кто внимательно читают "ZX-РЕВЮ", знают, что прямым путем в стандартном БЕЙСИКе это невозможно. Для этого мы набирали оператор THEN, а потом стирали его (см. "Маленькие хитрости" ZX РЕВЮ-91, стр.52). Это бывает полезно, если Вы работает в режиме: с отключенными токенами (KEYWORDS 4 см. ниже) или если Вы хотите ввести ключевое слово в строковую переменную, что бывает полезным при работе с командами REF, ALTER или KEYIN.

Управление вводом ключевых слов

Команда KEYWORDS позволяет Вам переключать режим ввода ключевых слов, т.е. вводить их одним нажатием клавиши, как в стандартном БЕЙСИКе, или набирать полностью по буквам, как это делается на компьютерах иных систем. Есть и режим KEYWORDS 3 в котором в одной строке можно одновременно применять и тот и другой подход. Это тоже может быть полезным. Даже если Вы хорошо знаете стандартную систему набора и Вам нравится набирать слова типа RANDOMIZE одним нажатием клавиши, все же ввести оператор IN по буквам несколько проще. Причем набор может идти как прописными, так и строчными буквами.

Ввод ключевых слов БЕТА-БЕЙСИКа

Есть два способа ввода ключевых слов БЕТА-БЕЙСИКа. Вы можете набирать их по буквам, используя ведущий пробел для того, чтобы отключить курсор "K", если необходимо, или можете использовать "одноклавишный" подход. В последнем случае команды и функции вводятся по-разному. Для ввода новой команды сначала перейдите в графический режим, а затем нажимайте соответствующую клавишу. Большинство клавиш в этом случае дают новые ключевые слова.

Для ввода новой функции наберите FN, а затем - "$" или "(" - в зависимости от того, что это за функция). Набирать "FN" можно теперь по-разному. Во-первых обычным порядком, как ключевое слово стандартного БЕЙСИКа, во вторых по буквам "f" + "n" + " " и, в-третьих, - нажав клавишу "Y" в графическом режиме.

Проверка синтаксиса

БЕТА-БЕЙСИК, как и стандартный БЕЙСИК, проверяет правильность того, что Вы вводите в компьютер и точно так же выдает звуковой сигнал BEEP, если устанавливает наличие ошибки. Изменить звуковой сигнал Вы можете, изменяя значения в ячейке памяти 23608 посредством POKE 23608,.. . Звуковой сигнал удобен, если Вы набираете программу,

например, из распечатки в журнале и при этом не часто смотрите на экран. Но имейте в виду, что если Вы работаете в режиме набора ключевых слов по буквам, то ошибки в их написании будут восприняты компьютером как ввод нового имени процедуры. Например, если вместо PAPER 1 Вы наберете PAPRE 1, то получите сообщение об ошибке "Missing DEF PROC" (отсутствует определение процедуры).

Команда LIST FORMAT

Эта команда позволяет Вам улучшить читабельность распечатки Вашей программы на экране монитора.

Команда CSIZE

Позволяет Вам уменьшить размер символов, доведя их количество в строке до 64 или, наоборот, увеличить их размер для печати заголовков или в иных, например в демонстрационных целях.

Команды JOIN <номера строк> и SPLIT.

Позволяют объединить две строки в одну или, наоборот, разбить строку на две.

Управляющий код "новая строка".

Вы можете ввести этот управляющий код в свою программную строку или в строку INPUT путем нажатия CAPS SHIFT и ENTER. В отличие от просто ENTER Вы сможете продолжать ввод той же программной строки на другой экранной строке. Смотрите также раздел "Управляющие коды" (CHR$ 15).

ГЛАВА 3. ПРОЦЕДУРЫ

Список используемых ключевых слов: DEF PROC, END PROC, LOCAL, DEFAULT, REF, READ LINE, LIST PROC и функция ITEM().

В этой главе мы кратко рассмотрим вопросы, связанные с применением процедур в БЕТА-БЕЙСИКе. Более подробную информацию Вы сможете получить далее, прочитав разделы относящиеся к каждому из указанных ключевых слов.

Использование процедур - весьма желанный прием для тех, кто программирует на БЕЙСИКе. Современные версии БЕЙСИКа в той или иной мере используют эту возможность и программисты встречают их с большим энтузиазмом. И, если Вы ими не пользуетесь, то наверное стоит пересмотреть свои взгляды, причем вовсе не потому, что это модно или потому, что это все рекомендуют, а просто так удобнее и легче писать и отлаживать программы.

БЕТА-БЕЙСИК 3.0 имеет одну из самых совершенных систем использования процедур среди других языков для домашних персональных компьютеров. Именно здесь Вы получаете наибольшую гибкость и эффективность программирования.

Основное преимущество использования процедур состоит в структурном программировании. Эта концепция предполагает, что Вы можете создать некоторый программный модуль, способный выполнять определенную работу и не оказывать никаких нежелательных побочных эффектов на остальную часть программы. Он не должен, например, изменять никакие переменные в программе, кроме тех, которые положено. После того, как этот блок программы вами написан и тщательно отлажен, Вы можете забыть о том, как он работает и из чего состоит. Когда Вы пишете новую программу и он Вам нужен, Вы просто пришьете его к тексту командой MERGE "", а используете - вызвав его по имени. Каждая процедура должна быть достаточно простой, чтобы быть вполне понятной. Если она у Вас получается громоздкой, сложной и непонятной, то стоит подумать о том, чтобы разделить ее на несколько логических частей и выразить каждую из этих частей своей процедурой. Процедура - это часть программы, имеющая свое имя и начинающаяся с оператора DEF PROC, после которого задается ее имя и имена тех переменных, с которыми она должна работать, а заканчивающаяся оператором END PROC.

Давайте рассмотрим простой, хотя и вполне бесполезный пример:

```
100 DEF PROC greet
```

```
110 PRINT "Hello"
120 END PROC
```

Попробуйте дать команду RUN и Вы увидите, что ничего не произойдет. Хоть процедура и часть программы, но работать в таком виде она не будет. Здесь мы записали только определение процедуры, то есть указали, что она должна делать. Выполнение процедуры происходит только после вызова ее по имени. Наша процедура имеет имя "greet". Попробуйте добавить строку

```
10 greet
```

Теперь, если Вы дадите команду RUN, в строке 10 будет запущена эта процедура и, в соответствии с ее определением, будет напечатано на экране слово Hello.

Все происходит точно так же, как в стандартном БЕЙСИКЕ с функциями. Там ведь тоже DEF FN игнорируется до тех пор, пока программа не встретит вызов функции оператором FN.

Вы можете столкнуться с проблемой того, как набрать слово greet в строке 10, ведь курсор находится в режиме "K", выше мы об этом упоминали:

- Вы можете нажать пробел и далее набирать greet по буквам;
- Вы можете дать команду KEYWORDS 4 и перейти в режим ввода всех ключевых слов по буквам;
- Вы можете использовать ключевое слово PROC и записать строку в виде:

```
10 PROC greet
```

Это совершенно то же самое, что и просто

```
10 greet
```

Использование ключевого слова PROC в данном случае просто дань привычке тех программистов, которые привыкли к такой записи по более ранним версиям БЕТА-БЕЙСИКА, хотя реально в нем больше нет никакой необходимости.

Далее мы будем говорить об использовании имени процедуры, как о вызове процедуры. Имя процедуры обязательно должно начинаться с буквы. Заканчивается имя процедуры пробелом, двоеточием, нажатием ENTER, операторами REF или DATA. Остальные символы, как правило, могут быть использованы в имени процедуры, но желательно, чтобы Вы привыкли использовать только буквы, цифры и символы подчеркивания - "_". Не имеет значения, какие буквы применяются строчные или прописные.

Определение процедуры начинается с DEF PROC и может быть расположено где угодно в программе. Его вполне можно располагать и до и после первого вызова процедуры. Важно только, чтобы оператор DEF PROC был первым оператором в строке. Определение процедуры может иметь сколько угодно строк и должно заканчиваться оператором END PROC.

Если Вы используете с одним DEF PROC несколько END PROC, то компьютер будет не в состоянии правильно "перепрыгнуть" через определение процедуры во время работы программы. В этом случае Вам придется самостоятельно позаботиться, чтобы он смог это сделать или размещать все лишние END PROC после оператора STOP. Изменим приведенный выше пример:

```
100 DEF PROC greet times
110   FOR n=1 TO times
120     PRINT "HELLO"
130   NEXT n
140 END FROC
```

В данном случае times - это параметр процедуры, он называется формальным параметром, поскольку не имеет пока никакого числового значения, а только указывает, как он используется при расчете процедуры. Раз он указан в определении процедуры, то он должен быть задан и при вызове процедуры. Теперь, когда вам надо будет исполнить процедуру greet, Вы зададите параметр times и заданное вами значение называется фактическим параметром - это число, которое фактически будет подставлено на место формального times при расчете процедуры.

Итак, формальные параметры - это просто имена, а фактические параметры - это числа, выражения или переменные (кроме тех случаев, когда используется REF, о чем см.

ниже).

Теперь, если Вы обратитесь к процедуре по имени, без задания параметра, например greet, то получите сообщение об ошибке "Variable not found" (переменная не найдена).

Если же Вы обратитесь к процедуре:

```
10 greet 5
```

Вы увидите что слово Hello будет пять раз напечатано на экране, т.к. фактический параметр "5" встал на место формального times.

Мы надеемся, что тем, для кого применение процедур в программах является новым делом, пока все понятно, но надо обсудить еще один очень важный момент, который может в больших программах повлечь за собой трудно обнаруживаемые ошибки.

Дело в том, что, как мы говорили, процедуры должны исполнять то, что записано в их определении и не должны оказывать нежелательных побочных воздействий на остальную часть программы. Рассмотрим нашу процедуру. Она манипулирует с двумя переменными times и n. Давайте теперь проверим, чему они равны после того, как процедура отработала. Дайте прямые команды:

```
PRINT times
PRINT n
```

Вы обнаружите, что times - не существует, а n - существует и имеет числовое значение.

Почему не существует times? Дело в том, что эта переменная введена нами в строке DEF PROC, что автоматически сделало ее локальной, то есть определенной только внутри процедуры. Когда процедура кончила работать, она удаляется из памяти и ее больше нет. И это хорошо, ведь если у Вас где-то еще в программе используется какая-нибудь переменная times, то ее значение могло бы быть изменено (испорчено), а теперь вам можно об этом не думать. А если у Вас до вызова процедуры уже была какая-то переменная times, то она тоже будет удалена? Нет, в момент вызова она будет запомнена, как глобальная переменная, а в процедуре будет создана локальная times. После работы процедуры локальная будет удалена, а глобальная - восстановлена. Хуже обстоит дело с переменной n, - она осталась существовать после работы процедуры и является глобальной, ведь она не входила в оператор DEF PROC. Ее значение может незаметно для Вас быть использовано где-то еще и привести к неприятным ошибкам. Для того, чтобы это не происходило, ее можно объявить локальной переменной внутри процедуры в принудительном порядке. Для этого служит оператор LOCAL.

Добавьте к нашей программе строку:

```
105 LOCAL n ,
```

а теперь добавьте к программе следующие строки, которые позволят вам убедиться, что после работы процедуры переменные times и n остались ненарушены:

```
10 LET n=1234
20 LET times=5678
30 greet 10
40 PRINT n,times
```

Процедура "greet" имела скорее учебное, чем практическое назначение, а вот более полезная процедура:

```
100 DEF PROC box x,y,width,height
130 PLOT  x,y: DRAW width,0
140 DRAW 0,-height: DRAW -width,0
150 DRAW 0,height
160 END PROC
```

Процедура "box" предназначена для рисования прямоугольника и имеет четыре параметра: x, y - координаты левого верхнего угла, width - желаемая ширина, а height - высота. Так, команда box 100, 100, 10, 40 изобразит вблизи центра экрану прямоугольник, вытянутый по вертикали.

Теперь предположим, что мы хотим, чтобы у нас был квадрат, а поскольку у него высота равна ширине, то попробуем ее не указывать, например:

```
box 100, 100, 50
```

Получим сообщение об ошибке, т.к. процедура имеет четыре формальных параметра, а мы им на смену подставили только три фактических. Это, однако, тоже можно

предусмотреть и предотвратить, для чего служит оператор default (по английски default - принятый "по умолчанию"). Запишем строку:

```
120 DEFAULT height=width
```

Эта строка буквально означает следующее: "Если параметр height не задан, то считать, что он равен параметру width".

Ну, а если Вы уж совсем ленивы, то можете не указывать и ширину квадрата, введя строку:

```
110 DEFAULT width=20
```

и у Вас и ширина и высота будут равны 20 пикселам. Используя запятые, можно опускать не только последние параметры, но и вообще любые.

```
box ,100,40,10
```

В этом вызове опущен параметр x. Разумеется, где-то в описании процедуры должен присутствовать оператор

```
DEFAULT x = ...
```

Передача параметров в виде ссылки

Итак, мы рассмотрели, как информация передается из главной части программы в процедуры с помощью параметров. Фактические параметры заменяют формальные, определенные в операторе DEF PROC. Но у нас может возникнуть необходимость не только передавать что-то в процедуру, но и, например, получать что-то от нее. Можно, конечно, написать процедуру, которая будет что-то рассчитывать, а результат расчета присваивать глобальной переменной x, из которой можно этот результат узнать после окончания работы процедуры и возврата в главную программу. Но это нарушит нашу договоренность о том, что процедура должна быть независимым модулем и не должна оказывать нежелательного влияния на другие участки программы (и на другие процедуры тоже). А если мы поступим таким образом с переменной x, то теперь должны будем все время помнить о том, что эту букву уже нигде нельзя использовать для иных целей, т.к. можно потерять то, что в этой переменной содержится. Желательно было бы уже при вызове процедуры и указании фактических параметров задать какую-то переменную, в которую надо поместить результат работы процедуры. Данная версия БЕЙСИКа, в отличие от многих аналогов, позволяет делать и это.

Обычно, при вызове процедуры в нее передаются параметры в виде значений (чисел), которые встают на место формальных параметров, но можно и передать просто имя переменной, без указания ее значения. Это называется передачей параметра, как ссылки, и делается добавлением оператора REF перед именем этой переменной в операторе DEF PROC. REF - это сокращение от английского слова reference (ссылка). Таким образом, переменная, которую Вы ставите при вызове процедуры в качестве фактического параметра, переименовывается в то имя, перед которым стоит REF, вычисляется в процедуре и возвращается под первоначальным именем в вызывающую программу или процедуру. В качестве демонстрации приведем процедуру SWOP, которая обменивает между собой содержимое двух строковых переменных.

```
200 DEF PROC SWOP REF a$, REF b$
210 LOCAL  t$
220 LET t$=a$: LET a$=b$: LET b$=t$
230 END PROC
```

А вызывается она, например, так:

```
10 LET x$="hi":LET y$="goodbye"
20 SWOP x$,y$
30 PRINT x$,y$
```

Без указания REF в определении функции и a$ и b$ тоже будут обмениваться своим содержимым в процедуре SWOP, но только именно внутри нее: глобального эффекта не будет, т.к. локальные a$ и b$ при выходе из процедуры будут утрачены. С использованием же REF эти переменные являются как бы временными именами, на которые ссылаются глобальные x$ и y$, поэтому изменения которым подверглись a$ и b$ в теле процедуры, отражаются и на x$ и на y$.

Языки программирования для микрокомпьютеров, использующие концепции

процедур, в большинстве случаев не позволяют использовать массивы в качестве параметров. БЕТА-БЕЙСИК разрешает это, но правда, требует, чтобы они передавались только как ссылки. Они переименовываются вместо того, чтобы просто копироваться в область локальных переменных процедуры. Этим достигается экономия памяти, ведь компьютеру не надо одновременно хранить один массив два раза. Если же вам на самом деле нужна локальная копия Вашего массива для каких-то временных манипуляций с ним, Вы можете внутри процедуры создать параллельный массив, объявив его как LOCAL, а затем скопировать в него содержание Вашего исходного массива, воспользовавшись для этого командой БЕТА-БЕЙСИКа COPY, о чем мы еще скажем ниже, когда будем ее рассматривать вместе с командой JOIN. Вот демонстрационный пример, в котором показано, как можно найти сумму элементов массива.

```
300 DEF PROC total REF a(),REF sum
310 LOCAL n
320 LET sum=0
330 FOR n=1 TO LENGTH  (1,"a()")
340 LET sum = sum + a()
350 NEXT n
360 END PROC
```

За именем массива должны идти скобки, чтобы интерпретатор отличал имена массивов от обычных переменных с тем же именем. Перед "sum" стоит оператор REF, так что по окончании работы содержимое "sum" будет передано глобальной переменной. Функция LENGTH (), о которой мы еще будем говорить, определяет размер массива для того, чтобы процедура могла работать с массивами любой длины.

Теперь зададим сам массив, чтобы убедиться, что наша процедура работает нормально:

```
100 DIM t(10)
110 FOR n=1 TO 10
120 LET t(n)=n
130 NEXT n
```

и добавим вызов нашей процедуры:

```
140 total t(), answer
150 PRINT answer
```

в итоге получим 55.

Передача параметров списком

Возможны варианты, когда вам вместо того, чтобы определять комбинацию параметров для процедуры, удобнее иметь дело со списком этих параметров, причем список может быть неопределенной длины. Чтобы это было возможным, в БЕТА-БЕЙСИКе 3.0 есть специальные средства.

Если в операторе DEF PROC использовать оператор DATA вместо обычного перечисления формальных параметров, то соответствующий ему оператор READ примет список фактических параметров, стоящих в вызове вашей процедуры. Чтобы эта возможность была по настоящему удобной, необходимо, чтобы можно было определить в процедуре есть ли еще параметры в списке, которые она не приняла. Для этого существует функция ITEM(). Она возвращает 0, если список исчерпан полностью, 1 если в списке есть не переданные параметры и следующий параметр - число, 2 - если следующий параметр - строковая переменная.

Приведен пример, который показывает, как можно организовать процедуру, которая просуммирует несколько заданных вами чисел.

```
100 DEF PROC SUM DATA
110 S = 0
120 DO UNTIL ITEM()=0
130 READ a
140 s = s+a
150 LOOP
160 PRINT sum
170 END PROC
```

Примененные в строках 120, 150 операторы DO UNTIL и LOOP - это удобная форма

организации цикла (см. далее). В принципе вместо DO UNTIL можно было бы применить и DO WHILE (см. ниже).

Вызов этой процедуры можно выполнить, например так:

sum 1, 2, 3, 4 или с других количеством параметров:

sum 1, 2, x, y, z, 1256

Если Ваш список параметров состоит из строковых переменных, то в строке 130 надо было бы применять READ a$, а не READ a. А как быть, если список смешанный и содержит и числа и строки? В этом случае перед READ надо проверить, очередной параметр с помощью ITEM() и использовать либо тот вариант, либо другой. В строковых переменных при этом можно избежать использования кавычек, если вместо READ использовать READ LINE (см. далее).

Обратите внимание на то, что список фактических параметров, передаваемых через DATA и READ, исключает возможность им быть локальными, если Вы специально это не зададите.

Рекурсия

Процедура, которая вызывает саму себя, называется рекурсивной. Есть анекдот, что в одном из компьютерных словарей-справочников написали:

РЕКУРСИЯ см. РЕКУРСИЯ.

Рекурсия часто помогает очень элегантно избегать сложных программистских проблем. С другой стороны, это не самая быстро работающая структура. К тому же возможны большие расходы памяти. Ведь при каждом вызове процедурой самой себя формируется новый временный список локальных переменных и для них выделяется память. Кроме того, несмотря на внешнюю простоту такой структуры, ее тщательный разбор может вызвать у программиста легкое головокружение. В качестве примера мы приведем процедуру, которая рисует бриллиант, а затем еще один такой же, но поменьше. Внутри первого и еще один внутри второго и так далее, на минимальный размер наложено ограничение, иначе процесс мог бы продолжаться бесконечно.

```
100 DEF PROC diamond x,y,size,diff
       DEFAULT diff=15
       PLOT x,y,-size
       DRAW -size,size
       DRAW size,size
       DRAW size,-size
       DRAW -size,-size
110 IF size >4 THEN
       diamond x,y+size,size-diff
       diamond x,y-size,size-diff
       diamond x-size,y,size-diff
       diamond x+size,y,size-diff
130 END PROC
```

Вызвать эту процедуру можно, например, так:

```
diamond 128,88,40
```

Ошибки

Если Вы попробуете вызвать процедуру, которую забыли задать, Вам дадут сообщение об ошибке W: "Missing DEF PROC". Если же Вы забудете закрыть процедуру с помощью END PROC - сообщение X: "No END PROC". Программа будет стараться во время работы "перепрыгнуть" через блок, в котором задается процедура и не сможет этого сделать.

Если при вызове задано больше фактических параметров, чем их есть в наличии в описании процедуры, - сообщение "Parameter error." Если тип формальных параметров не совпадает с типом реально установленных фактических параметров - "Nonsense in BASIC". Оператор END PROC может генерировать сообщение "Variable not found", если оказывается, что переменная, которую процедура должна передать в качестве выходного параметра, не существует.

ГЛАВА 4. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Кроме заложенной в БЕТА-БЕЙСИК 3.0 концепции использования процедур, о чем мы только что написали, введены еще дополнительно некоторые средства, обеспечивающие возможность структурного программирования: операторы DO,LOOP, EXIT IF, WHILE и UNTIL обеспечивают те же дополнительные возможности организации циклов, что и REPEAT и WHILE в более ранних версиях еще больше повышают гибкость в программировании.

В структуре операторов IF…THEN можно использовать ELSE.

Оператор ON позволяет выбрать необходимый номер строки. Это как бы упрощенная форма операторов CASE или SWITCH.

Команда LIST FORMAT может вам обеспечить вывод текста программы в структурированной форме ("лесенкой") так, как это принято в языках, поддерживающих структурное программирование (ПАСКАЛЬ, СИ и др.).

ГЛАВА 5. ОБРАБОТКА ДАННЫХ

В данной версии языка введены дополнительно следующие операторы обеспечивающие обработку данных разного типа:

JOIN, COPY, DELETE, SORT - для работы с массивами и со строковыми переменными.

INARRAY и INSTRING - поисковые функции.

Функции LENGTH, CHAR, NUMBER.

Операторы: EDIT <переменная>, SAVE DATA, USING.

Функции: EOF, SHIFT$, USING$.

JOIN и COPY позволят вам перемещать или копировать массив данных или его часть в другой массив. Теперь Вы можете во время работы изменять размер массива. Оператор DELETE позволяет удалить элемент массива, а SORT - выполнить его сортировку по алфавиту или по числу. Эти же команды могут быть использованы и при работе со строковыми переменными.

Функция INARRAY выполняет просмотр массива и поиск в нем нужного Вам элемента, то же самое делает INSTRING, но для строковых переменных. Функция LENGTH выдает размер массива и его месторасположение. Она может позволить Вам разделить массив на части и загрузить и обработать его по частям, если он слишком велик, чтобы поместиться в памяти компьютера в то время, как там присутствует БЕТА-БЕЙСИК 3.0. Функции CHAR$ и NUMBER дают возможность создавать "целые" массивы.

Теперь Вы можете редактировать (изменять) переменные в той же мере, как Вы редактируете программные строки. Все программные переменные можно отгружать на ленту единым блоком с помощью SAVE DATA. Форматирование данных можно выполнить с помощью USING или USING$. Функция EOF (End Of File - "конец файла") служит для работы с микродрайвом и может сигнализировать о том, что ввод данных из файла завершен. Оператор SHIFT среди прочих дел выполняет и такую полезную операцию, как изменение регистра букв, которыми записана строковая переменная.

ГЛАВА 6. ГРАФИКА

Используются следующие операторы и функции:

ALTER, CONTROL, CODES, CSIZE, DRAW TO, FILL, GET <Область экрана>, OVER 2, PLOT, POKE, ROLL, SCROLL, WINDOW, XOS/XRG/YOS/YRG, функции SINE, COSE, FILLED, MEMORY$, SCRN$.

Вот в двух словах, для чего они предназначены (более подробно мы рассмотрим каждый оператор и каждую функцию в ближайших выпусках):

ALTER - позволяет гибко управлять цветовыми атрибутами экрана.

DRAW TO - вычерчивание линий к заданной координате.

GET - сохраняет заданную область экрана в виде строковой переменной.

PLOT - восстанавливает на экране (в произвольной области) сохраненный с помощью GET <фрагмент>.

CSIZE - с его помощью Вы можете увеличить или уменьшить размер фрагмента экрана, принятого с помощью GET перед тем, как восстанавливать его по PLOT.

POKE - допускает быстрые манипуляции с областями памяти.

FILL - заполняет область экрана, находящуюся внутри замкнутого контура, избранным цветом INK или PAPER.

ROLL - перемещение экрана или его части в заданном направлении.

SCROLL - то же самое, но с возвратом, когда например то, что ушло за левую границу экрана, начинает появляться справа.

SCRN$ - распознает символы графики пользователя.

WINDOW - организация концепции "окон".

XOS, XRG, YOS, YRG - изменяет исходную координату экрана для графических функций и масштабный коэффициент по двум направлениям.

ГЛАВА 7. СЕРВИСНЫЕ И ОТЛАДОЧНЫЕ ВОЗМОЖНОСТИ

ALTER (...) - поиск и замена по тексту программы.

AUTO - автоматическая нумерация строк.

DEF KEY - этой командой Вы можете задать до 36 операторов, функций или строковых сообщений на пользовательских клавишах.

DELETE - удаление блока программных строк.

LIST/LLIST - распечатка текста программы или ее фрагмента в диапазоне от...до.

LIST/LLIST DATA/VAL/VAL$ распечатка программных переменных.

LIST/LLIST DEF KEY - распечатка определений, присвоенных пользователем назначенным клавишам.

LIST/LLIST PROC - распечатка текста процедуры.

LIST/LLIST REF распечатка только тех строк, в которых есть оператор REF.

REF - поиск программных строк с этим оператором.

RENUM - перенумерация программного блока или копирование.

MEMO - эта функция возвращает доступный объем свободной памяти.

(Продолжение следует)

МИХАЙЛЕНКО В.С.

ЗАЩИТА ПРОГРАММ

Сегодня мы начинаем печатать объемный труд нашего читателя из г. Минска Михайленко В.С., посвященный вопросам защиты компьютерных программ. Книга написана им специально для "ZX-РЕВЮ".

Мы очень рады, что лед тронулся и стали появляться крупные работы отечественных авторов. За те несколько лет, что мы работаем в области информационного обеспечения "Синклер"-совместимых машин, мы очень хорошо узнали, как много у нас талантливейших программистов, способных разработчиков. Но с другой стороны, мы хорошо знаем и то, как трудно их подвинуть на то, чтобы написать хорошую книгу, статью, цикл статей. "Сделаю все, что угодно, но писать книгу - разве что под дулом пистолета", - вот типичный ответ нашего разработчика. Значительно меньшая часть все-таки соглашается, причем охотно, но после первого порыва быстро охладевает и на этом все кончается.

Откуда же мы знаем о существовании этой армии талантов, если они ничего не пишут? - может спросить внимательный читатель. Оказывается, все-таки пишут. Они пишут интересные, хорошо проработанные, наполненные точным, конкретным содержанием критические письма на то, что читают в "ZX-РЕВЮ". Мы благодарны им за это, но увы должны отметить, что собраться и написать пять-шесть страниц, блещущих идеями, нашему разработчику проще, чем собраться с духом и подготовить нормальный законченный материал, освещающий хотя бы одну идею, и который с интересом и пониманием будут читать миллионы начинающих поклонников "Спектрума", а именно такой цифрой мы и оцениваем на сегодняшний день тех, кто уже включился в это всенародное движение.

Итак, мы предоставляем страницы В.С.Михайленко для освещения полезного, как нам кажется, вопроса защиты компьютерных программ. Мы полагаем, что кроме своей утилитарной цели то, о чем он пишет, имеет еще и большое методологическое значение, поскольку приоткрывает перед Вами завесу над многими вопросами, связанными с системными особенностями "Спектрума".

Работа прошла техническое и литературное редактирование "ИНФОРКОМа".

ЧАСТЬ 1

Система защиты компьютерных программ представляет собой исключение возможности просмотра текста программы и внесение в него изменения лицом, не уполномоченным на это автором программы.

Для компьютера "ZX SPECTRUM" существует система мер по защите "Бейсиковских" программ и программ в машинных кодах. Этот цикл статей посвящается защите Бейсик-программ.

Методов защиты программ существует очень много, но среди них можно выделить ряд направлений, по которым можно осуществить подход к данной проблеме:

1. Исключить возможность остановки (прерывания работы программы).
2. Сделать листинг программ нечитаемым.

ГЛАВА 1. Исключение возможности остановки программ.

1.1. Общие рекомендации.

Самое первое, что необходимо сделать - это выполнить программу автостартующей со строки с номером n. Это выполняется при записи программы на ленту командой:

```
SAVE "имя" LINE n
```

Теперь после загрузки программа сама будет стартовать с указанной строки. Тем не менее, она может быть остановлена командой BREAK или по сообщению об ошибке.

Отключить клавишу BREAK можно, если в той строке, с которой происходит автостарт, поместить команду POKE 23613, PEEK 23730-5. Кроме этого, возможно отключение клавиши BREAK за счет использования подпрограммы в машинных кодах, которая будет описана

ниже.

Если же Вы по каким-то причинам не желаете использовать подпрограммы в машинных кодах то необходимо помнить следующее.

Для хорошо отлаженной программы остановку по ошибке можно исключить. Надо знать, что чаще всего при взломе программы сознательно вносят ошибку во время исполнения оператора INPUT. Например, если программа просит от пользователя ввести какое-либо число, он сознательно вводит букву. Программа останавливается с сообщением VARIABLE NOT FOUND (переменной нет). Чтобы избежать этого рекомендуется не использовать оператор INPUT, а организовывать опрос клавиатуры на основе функции INKEY.

Если же Вам необходимо использование оператора INPUT (например при вводе многозначных чисел), то с целью защиты от прерывания можно сделать, чтобы программа либо "зависала" (или самосбрасывалась при остановке), либо вместо сообщения об ошибке осуществляла переход на заранее заданную строку.

Самосброс программы дает использование нижеприведенных команд. Так, если поместить их в строке автостарта, то при попытке сделать BREAK или появлении сообщения об ошибке программа будет сбрасываться:

```
LET ERROR=256*PEEK 23614+PEEK 23613: POKE ERROR, 0: POKE ERROR+1, 0
```

Для обеспечения зависания программы используется размещение в стартовой строке команды:

```
POKE 23659, 0
```

Однако, пользователей компьютера ZX SPECTRUM наверняка заинтересует напомненный им еще раз факт: ни в одной фирменной программе (имеется в виду рабочая программа, а не загрузчик) не происходит сбрасывания или зависания при нажатии на клавишу BREAK. Если после длительной загрузки программа зависнет или сбросится, то это будет иметь недружественный эффект. Гораздо эффективнее в этом плане использование специальных программ в машинных кодах, которые вместо выдачи сообщения об ошибке осуществляют переход на заранее заданный шаг программы.

Эти программы и методы их использования описаны в следующей статье.

1.2 Подпрограмма обработки сообщений с кодами D, H, L.

Нажатие клавиши BREAK во время исполнения программы (или же ей аналогичное STOP IN INPUT) в обычном режиме может приводить к одному из следующих сообщений:

`D: BREAK - CONT REPEATS`

Клавиша BREAK нажата во время действия периферийной операции. Действие CONTINUE после этого оператора обычное, т.е. то, что указано в операторе.

`H: STOP IN INPUT`

Некоторые введенные данные начинаются с оператора STOP, или была нажата INPUT LINE.

Действие CONTINUE обычное.

`L: BREAK INTO PROGRAM (BREAK во время исполнения программы)`

Нажата клавиша BREAK; это было обнаружено между исполнением двух операторов. Строка и номер оператора в строке указывают на оператор, выполненный перед нажатием BREAK, но CONTINUE переходит к следующему оператору.

Чтобы во время нажатия клавиши BREAK остановки не произошло, может быть использована, например подпрограмма в машинных кодах ON BREAK GO TO (подпрограмма N66 из инструментального пакета SUPERCODE 3.5).

Если Вы запустите эту подпрограмму со строки автостарта, то типы остановок D,H,L будут игнорироваться, а программа будет переходить к заранее заданному Вами номеру строки (Первоначально задана строка 9495).

Для тех, кто не располагает пакетом SUPERCODE 3.5, мы приводим дисассемблер данной процедуры.

```
              ORG 60899
60899 CD7C00  CALL 0124
60902 3B      DEC SP
```

```
60903 3B         DEC SP
60904 E1         POP HL
60905 010F00     LD BC,15
60908 09         ADD HL,BC
60909 EB         EX DE,HL
60910 2A3D5C     LD HL,(23613)
60913 73         LD (HL),E
60914 23         INC HL
60915 72         LD (HL),D
60916 C9         RET
60917 76         HALT
60918 CD8E02     CALL 654
60921 7B         LD A,E
60922 FEFF       CP 255
60924 20F8       JR NZ,60918
60926 3A3A5C     LD A,(23610)
60929 FE0C       CD 12
60931 280A       JR Z,60943
60933 FE10       CP 16
60935 2806       JR Z,60943
60937 FE14       CP 20
60939 2802       JR Z,60943
60941 1819       JR 60968
60943 3C         INC A
60944 32815C     LD (23681),A
60947 FD3600FF   LD (IY+0),255
60951 211725     LD HL,9495         ; загрузка номера строки перехода в Бейсике.
60954 22425C     LD (23618),HL
60957 210000     LD HL,0
60960 22445C     LD (23620),HL
60963 3B         DEC SP
60964 3B         DEC SP
60965 C37D1B     JP 7037
60968 C30313     JP 4867
```

Наилучшим вариантом было бы, если бы Вам удалось набрать эту процедуру в Ассемблере и, откомпилировав ее, получить соответствующую программу в кодах. Для тех, кто не имеет такой возможности, привожу программу на Бейсике, которая введет и запустит данную процедуру.

```
5 CLEAR 24999
10 FOR 1=25000 TO 25071
20 READ A: POKE I,A
30 NEXT I
40 DATA 205,124,0,59,59,225,1,15,0,9,235,42,51,92,115,35,114,201,118,205,142,2,
     123,254,255,32,248,58,58,92
50 DATA 254,12,40,10,254,16,40,6,254,20,40,2,24,25,60,50,129,92,253,54,0,255,33,
     23,37,34,66,92,33,0,0,34,68,92,59,59,195,125,27,195,3,19
```

Данная Бейсик-программа позволяет вводить эту подпрограмму в любое место оперативной памяти. Теоретически возможны 3 варианта:

1) Ваша Бейсик-программа сама формирует эту процедуру и запускает ее, т.е. вышеприведенная Бейсик программа является началом Вашей исходной программы, которую необходимо защитить.

2) Вы формируете в удобном месте эту процедуру в кодах, записываете ее на кассету в виде отдельного файла машинных кодов и потом, с помощью программы-загрузчика загружаете эти коды и запускаете.

3) Вы совмещаете программу на Бейсике и эту программу в кодах с использованием оператора REM методом который будет описан в одной из следующих глав и затем сразу после загрузки, непосредственно из Бейсика запускаете эту процедуру.

Третий метод является самым удобным. К недостаткам первого относится то, что пока Бейсик будет формировать подпрограмму в кодах (перебрасывать числовой массив),

взломщик уже сумеет остановить программу. Второй метод усложняет загрузку: в самом деле, Вам придется делать программу загрузчика, загружать коды, запускать их и лишь потом, обезопасив себя от взлома, загружать свою основную программу.

1.3. Подпрограмма обработки сообщений об ошибке, кроме 0:OK; 8:END OF FILE; 9:STOP STATEMENT.

Многие программы имеют возможность ввода ошибки, т.е. останавливаются из-за неправильной своей работы. В этом случае на экране дисплея индицируется сообщение об ошибке. Чтобы этого не происходило, а вместо выдачи сообщений осуществлялся переход к заранее заданной строке Бейсик-программы, используется процедура: ON ERROR GO TO (процедура N 65 из программы SUPERCODE).

Ниже приведена Бейсик-программа, с помощью которой можно загрузить процедуру : "ON ERROR GO TO" в любое место оперативной памяти, в моем примере загрузка осуществляется с адреса 25000.

```
10 CLEAR 24999
20 LET A=25000
30 FOR I=0 TO 72
40 READ B
50 POKE A+I, B
60 NEXT I
70 REM ЗАПУСК ПРОГРАММЫ
80 RANDOMIZE USR 25000
90 DATA 205,124,0,59,59,225,1,15,0,9,235,42,61,92,115,35,114,201,59,59,205,142,2,123,
      254,255,32,248,58,58
100 DATA 92,254,255,40,33,254,1,40,29,254,8,40,25,60,50,129,92,253,54,0,255,33,23,37,
      34,66,92,175,50,68
110 DATA92,253,203,7,254,195,125,27,51,51,195,3,19
```

Процедура "ON ERROR GO TO" имеет длину 73 байта и осуществляет переход к строке Бейсика 9495 (допустимы изменения).

Рассмотрим, для чего же употребляется эта программа. В разделе 1.1. было сказано, что одним из методов взлома является ввод несуществующей переменной вместо числа в операторе INPUT. На уровне Бейсика рекомендовалось использовать опрос клавиатуры с помощью INKEY$. Данная процедура самостоятельно анализирует подобного рода ошибки и осуществляет переход к строке 9495. При этом по адресу 23681 можно узнать код ошибки.

Точно такой же переход эта программа будет осуществлять, если будет произведена попытка деления на 0. Оператор PRINT попытается распечатать символ с помощью управляющего кода AT за пределами экрана; встретится RETURN без GO SUB или NEXT без FOR и др.

Наилучшим способом использования данной процедуры является совмещение ее с Бейсик-программой с использованием оператора REM методом, который будет ниже.

Чтобы подготовиться к его выполнению, Вам необходимо сформировать эту программу в произвольной области ОЗУ, используя Бейсик-программу данного раздела и записать ее на магнитофон в виде отдельного файла CODE.

1.4. Метод защиты, основанный на совмещении Бейсика и программы в машинных кодах.

Многие пользователи компьютера ZX SPECTRUM бывают удивлены, когда после загрузки всего лишь одного Бейсик-файла программы на экране возникают эффекты, которые на Бейсике создать заведомо невозможно. Такое же чувство возникает и тогда, когда после загрузки первого Бейсик файла дальнейшая загрузка ведется необычным способом, несмотря на то, что загружался именно Бейсик-файл (свидетельством этого является то, что Вы подали команду LOAD "").

Все дело в том, что в данном случае использован прием, позволяющий совместить в первом загружающемся файле как Бейсик-команды, так и машинные коды.

Рассмотрим, как это осуществить на практике. Предположим, у нас имеется

отлаженная программа в машинных кодах, имеющая длину n байтов, и мы желаем совместить ее с Бейсик-программой (Данное совмещение накладывает ограничение на программу в кодах: она должна использовать индексную адресацию, переходы могут быть только относительными, т.е. необходимо, чтобы эта программа работала, будучи загруженной в любое место оперативной памяти).

Один из приемов состоит в том, чтобы разместить программу в кодах после оператора REM, т.к. текст, следующий после этого оператора, не обрабатывается компьютером. (Подобное совмещение также накладывает ограничение на использование кодов в программе. Ни один код этой программы не должен быть равен 13(DEC), 0D(HEX), поскольку в Бейсике этот код свидетельствует об окончании строки и символы, следующие за ним, обрабатываются как новая строка Бейсика).

Для того, чтобы осуществить совмещение, нам необходимо сформировать строку с оператором REM так, чтобы количество символов, следующих после REM, было равно числу байтов программы в кодах, т.е. n.

Для этого наберем с клавиатуры:

```
1 REM LLLL...LLLL
```

Мной был использован символ L но в принципе можно использовать к любой другой.

Как Вы уже обратили внимание, строка с оператором REM должна идти первой в программе. Это необходимо для того, чтобы легко можно было определить адрес, по которому нужно грузить машинные коды и из которого вызывать их на исполнение.

Если Вы не изменяли системную переменную PROG, то она указывает на начало Бейсик программы по адресу 23755.

Следовательно, загрузку кодов нам необходимо начинать с адреса 23760 (Это объясняется тем, что первые два кода Бейсик-строки характеризуют ее номер, следующие два - ее длину и пятый символ - это код оператора, в данном случае REM).

Теперь заменим набранные нами символы в количестве n байтов на программу в машинных кодах аналогичной длины. Подадим команду

```
LOAD "" CODE 23760, N
```

(В случае, если системная переменная PROG у Вас смещена, то Вам необходимо узнать адрес начала загрузки кодов, подав следующую команду с клавиатуры:

```
PRINT (PEEK 23635+256 * PEEK 23636+5)
```

Теперь, при загрузке кодов, вместо адреса 23760 укажите адрес, который Вы получили. Загрузив коды, мы добились поставленной цели. Теперь в Бейсике находится программа в кодах, которую при желании можно запустить, подав в одной из нижеследующих строк Бейсик-программы, команду

```
RANDOMIZE USR 23760
```

В случае смещенной системной переменной PROG, Вам будет необходимо видоизменить команду:

```
RANDOMIZE USR (PEEK 23635+256 * PEEK 23636+5)
```

Метод совмещения чаще всего используется, когда Вы хотите защитить наиболее уязвимые места программы, которые Вам наиболее дороги и Вам бы не хотелось, чтобы они были кем-либо изменены. Кроме этого, такой подход позволяет загружать машинные коды без "заголовка" методом, который будет описан далее, что затрудняет вскрытие данного машиннокодового блока, поскольку не известен его адрес.

Как Вы уже обратили внимание, обычно загружающийся программный блок состоит из двух частей. Первая часть является тем заголовком, который содержит название программы, ее длину и адрес, начиная с которого она располагается в памяти компьютера, а вторая часть - это уже непосредственно коды. Следует отметить, что аналогичную структуру имеет и Бейсик программа и массивы, но "заголовки" этих блоков имеют принципиальные различия.

Среди наиболее любопытных особенностей метода скрывания машинного кода в строке REM следует отметить тот факт, что после подачи команды LIST распечатка может быть остановлена с сообщением о неправильном цвете (INVALID COLOR). Это объясняется тем, что в машинных кодах могут встретиться команды, код которых не может быть опознан компьютером в строке Бейсик-программы. И этот любопытный факт, совместно с

использованием метода зануления, который будет описан в следующей главе, дает великолепную защиту от листинга. Необходимо, однако, предупредить читателя, что создавая программы в кодах, для их последующего размещения в Бейсике, необходимо сразу определиться в адресах, т.е. точно установить адреса, где данная программа будет находиться и работать, чтобы не было сбоев из-за неправильной адресации.

Если Вы все же хотите использовать программу в кодах, которая была рассчитана на определенное место в памяти, то чтобы ничего не переделывать, Вам просто необходимо дополнить ее размещенным в начале блоком переноса с использованием команды LDIR (более подробное описание этой команды дано в соответствующих методических разработках "Инфоркома").

1.5. Оригинальный метод защиты, использующийся при загрузке машинных кодов.

Самые первые программы для ZX SPECTRUM отличались примитивностью в плане защиты (да и не только защиты). Обычно они состояли из Бейсик-программы, управляющей загрузкой, которая загружала коды и запускала их с определенного адреса. Взломщик мог беспрепятственно смотреть как Бейсик программу, так и программу в кодах.

Более поздние авторские разработки были гораздо лучше продуманы в отношении защиты.

Как известно, загружающаяся программа состоит из заголовка и самой программы. В заголовке находится описание типа программы: Бейсик, CODE, SCREEN$ и т.д. Кроме этого, там находится название программы и адрес в памяти, с которого начнется загрузка, а также длина программы (для программ в машинных кодах).

Чтобы скрыть эти данные, т.е. длину и адрес начала, разработчики создали систему загрузки, позволяющую загружать машинные коды без заголовка.

Это делается с использованием специальной процедуры в кодах, которая в некоторых случаях совмещается с программой на Бейсике с использованием оператора REM.

К программам, использующим данный метод загрузки, относятся MAT II, RAMBO III и др.

Рассмотрим, как этот прием реализован программе MAT II. Ниже приведен листинг загрузчика.

```
0 REM CRACKED BY MIHAILENKO VS 1991
10 FOR n=0 TO 12: READ a: POKE 64000+n, a
20 NEXT n
30 DATA 55, 62, 255, 221, 33, 0, 64, 17, 254, 27, 195, 86, 5
40 RANDOMIZE USR 64000
```

Фактически Бейсик-программа служит лишь для того, чтобы сформировать и запустить процедуру в машинных кодах. Запуск осуществляется в строке 40, а строки 10...30 ответственны за формирование этой процедуры, начиная с адреса 64000. Ниже приведен дисассемблированный текст данной процедуры.

```
64000 SCF
64001 LD A,255
64003 LD IX,16384
64007 LD DE,7166
64010 JP 1366
```

Действует эта программа следующим образом. В регистр IX загружается адрес начала загрузки кодов, в регистр DE - общая длина программы в кодах, остальная система команд необходима для правильной работы встроенной в ПЗУ программы загрузчика с ленты. Переход на эту программу осуществляется в строке 64010. (Все используемые величины - десятеричные).

Этот тип загрузки используется во многих программах. В частности, программа GREEN BERET использует встроенную процедуру загрузки как подпрограмму, вызывая ее по

CALL 1366 для загрузки нескольких блоков кодов.

Следует добавить, что аналогичным образом действует загрузка в программе RAMBO III. Отличительной ее особенностью является загрузка кодов, занимающих всю оперативную память. Никаким другим способом здесь загрузку осуществить бы не удалось.

1.6. Метод защиты, используемый в программе FIST III.

Защита, используемая в программе FIST III, достаточно проста и не является идеально надежной. Но, в то же время, об этой системе команд мало кто знает и поэтому в некоторых случаях она может оказаться достаточно эффективной.

По нашей классификации этот тип защиты относится к разряду тех, которые делают листинг программы нечитаемым. Для этих целей бордюр окрашивается в черный цвет и подается система команд:

```
BORDER 0: POKE 23624, 0: POKE 23570, 16
```

Этот метод защиты легко можно разблокировать, подав команды:

```
BORDER 7: POKE 23570, 6
```

Кроме защиты, эта программа имеет еще одну особенность, на которую могли не обратить внимание пользователя: заставка в середине экрана формируется на Бейсике. Создана она довольно оригинальным способом, но самым любопытным является то, что системная переменная, используемая в этой программе, в фирменном описании компьютера (автор - Виккерс) охарактеризована, как неиспользующаяся.

Ниже приведем текст программы, после выполнения которой в середине экрана очень крупным шрифтом появляется надпись: GOOD LUCK TO YOU

```
10 REM CRACKED BY MIHAILENKO VADIM MINSK 1991
20 FOR I=72 TO 79
25 POKE 23681, I
30 LPRINT "GOOD LUCK TO YOU"
40 NEXT I
```

В данном случае авторы используют команду вывода на печатающее устройство LPRINT для поэтапного высвечивания на экране вышеуказанной надписи.

Если Вы хотите получить подобное изображение не в середине, а в верхней части экрана, то Вам необходимо подобрать соответствующим образом изменение переменной I.

Любопытный эффект получается при использовании изменения

```
20 FOR I=72 TO 78 и т.д.
```

ГЛАВА 2. Методы защиты от листинга или как сделать текст программы нечитаемым.

Данная глава посвящается изучению методов защиты от листинга, применяемых в большинстве фирменных программ, в частности в их загрузчиках, написанных на Бейсике, либо так или иначе использующих Бейсик. Кроме этого, рассмотрены некоторые методы, применяемые "взломщиками" компьютерных программ, т.н. крэккерами, в частности Билом Гилбертом (Bill Gilbert - если человек с таким именем действительно существует, а не использует псевдоним); PEGAZ SOFTWARE (т.н. корпорация по взлому, которая даже оставляет свой телефон); ROBY CRACKING SERVISE, использующем в своем арсенале достаточно примитивные приемы в отличие от того же Билла Гилберта и пр.

В связи с этим я обращаюсь ко всем читателям данного материала с просьбой не использовать описанные мной приемы взломщиков для установки в созданных не Вами программах Ваших надписей и т.п., поскольку это противоречит элементарным этическим нормам, не говоря уже о порядочности.

После длительной работы я могу сказать с уверенностью, что в компьютере ZX SPECTRUM нельзя создать полностью защищенных программ. Это налагает большую ответственность на человека, описывающего приемы взломщиков. Я ничего не скрывал в изложении данного материала, полагаясь на порядочность и внутреннюю культуру читателей.

Призываю Вас использовать данные приемы исключительно для защиты своих программ, а также для совершенствования своей техники программирования. Вы можете использовать описанные ниже рекомендации для входа в программу с целью изучения новых приемов программирования, но к тем, кто использует, их для того, чтобы калечить чужие программы, не умея написать свои, надо относиться также, как мы отнеслись бы к человеку, который на наших глазах лазит по чужим карманам.

2.1. Общие рекомендации.

Простейший прием сокрытия листинга состоит в том, чтобы сделать одинаковыми цвета символов (INK) и фона (PAPER).

В тех местах, где программа должна сделать вывод на экран в операторе PRINT вставляют в качестве временных правильные цвета.

Например, подав вначале программы команды:

```
10 INK 7: PAPER 7
```

по мере необходимости используем нормальный цвет:

```
20 PRINT INK 0; "ZX SPECTRUM"
```

Другой прием состоит в искажении набора символов путем задания "фальшивого" значения системной переменной CHARS. Для примера наберем с клавиатуры.

```
POKE 23606,8: PRINT "ZX SPECTRUM": POKE 23606,0.
```

* * *

Небольшое отступление:

Краткое описание системной переменной CHARS.

Как известно, системная переменная CHARS ответственна за место расположения шрифта в ПК "SPECTRUM". В своем обычном содержании она указывает на адрес шрифта, зашитого в ПЗУ, а точнее - на тот адрес, который находится на 256 байтов ниже. Это легко проверить:

```
PRINT PEEK 23606
PRINT PEEK 23607
```

Точный адрес можно узнать, подав с клавиатуры команду:

```
PRINT PEEK 23606 + 256*PRINT PEEK 23607+256
```

Вы получите 15616. При этом по адресу 23606 находится 0, а по адресу 23607 - число 60.

Среди практических использований этой системной переменной следует отметить возможность создания своих шрифтов в т.ч. русского и других национальных, а также переключение шрифтов с одного на другой. Подробно эта технология разобрана в разработке "Большие возможности Вашего "СПЕКТРУМа", выпущенной ИНФОРКОМом в 1989 г.

В окончании комментария привожу любопытный пример использования мною этой системной переменной. Многие из Вас, вероятно, видели, что при загрузке некоторых фирменных программ используются счетчики. Они могут быть различны. Либо это постоянно укорачивающаяся линия (уменьшение длины свидетельствует о процессе загрузки, а сама длина показывает непосредственно сколько загрузилось и сколько осталось), использованная, в частности в программе SPLITTING, а также цифровые счетчики, имитирующие механические, подобные тем, что показывают пробег в автомобиле.

Кроме того, такая надпись, имитирующая механический счетчик, использована во время демонстрации в программе FIST III (режим DEMONSTRATION, который появляется после загрузки, если не нажимать никакие клавиши).

Подобный имитатор механического счетчика можно создать и Вам, если использовать возможности системной переменной CHARS.

Наберете предложенную ниже программу и запустите ее:

```
10 FOR I=80 TO 0 STEP -1
20 POKE 23606, I: PAUSE 10
30 PRINT AT 11,15;"0"
40 NEXT I
```

Подобный очень любопытный эффект наблюдается потому, что за каждый цикл

операторов FOR-NEXT процедурой, ответственной за печать, распечатываются 8 байтов предполагаемого знака 0 (в данном случае). Но с каждым проходом системная переменная все ближе подходит к своему истинному значению, а за счет того, что шаблоны цифр в ПЗУ расположены рядом, создается впечатление, что цифры следуют одна за другой по возрастанию или убыванию.

Схематически этот процесс можно представить так:

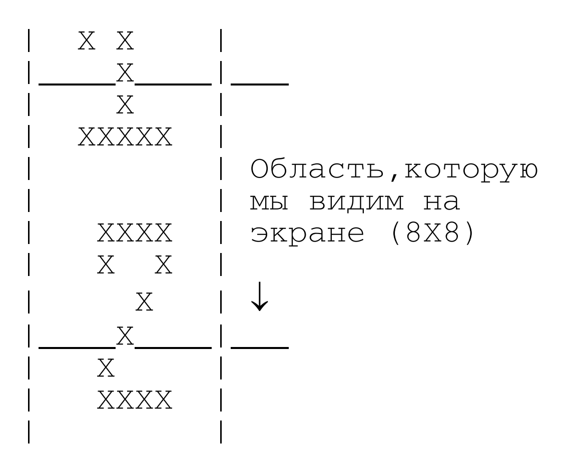


Рис.1

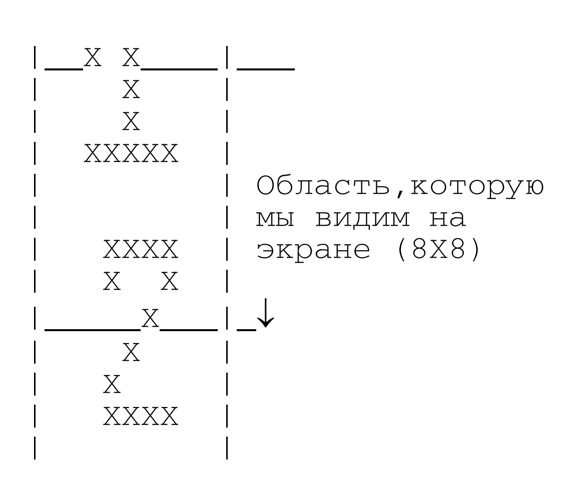


Рис.2

В этой схеме горизонтально расположены байты шаблонов символов, хранящиеся в ПЗУ, и, в зависимости от того, на какой адрес указывает системная CHARS, в области, появляющейся на экране (на экране появляется одно знакоместо 8X8), мы видим опускающиеся вниз цифры, имитирующие движение механического счетчика, до тех пор, пока они не достигнут необходимой нам величины - в моем примере 0.

* * *

Итак, системная переменная CHARS может быть использована для защиты от листинга в Бейсик программах. Если адресовать ее в те области памяти, где вообще ничего нет, например, POKE 23607,200 , то все символы будут выглядеть, как пробелы и на экране вообще ничего не будет. Вам же необходимо перед всяким оператором PRINT включать нормальный режим и выключать его после PRINT.

2.2. Универсальная система защиты - метод нулевых строк.

Почти во всех программах к ZX SPECTRUM присутствует нулевая строка. В большинстве программ она выполняет следующие функции:

1) Защита машинных кодов, размещенных после оператора REM от редактирования.

2) Защита от листинга той части Бейсика, который размещен в нулевых строках.

3) Защита от редактирования надписей, оставляемых фирмами и взломщиками.

Для того, чтобы сделать первую строку программы нулевой, необходимо подать команды:

```
POKE 23755,0
POKE 23756,0
```

Образующаяся строка имеет номер 0. Она реально обрабатывается программой но не поддается редактированию, т.е. ее невозможно вызвать командой EDIT.

Предлагаемая Вашему вниманию универсальная система защиты включает в себя "зануление" всех строк программы (разумеется, этот метод действенен только для программ, где отсутствуют команды условных и безусловных переходов - GO TO, GO SUB и наибольшую эффективность приобретает для небольших программ, управляющих загрузкой). Это становится возможным, т.к. программа выполняет все команды последовательно, а номера строк использует только для команд-переходов. Преимуществ же у этого метода довольно много: при удачном сочетании его с методом защиты от листинга, использующим управляющие коды (подробно описан в следующей статье), это позволяет сделать текст Вашей программы полностью нечитаемым.

Действительно, абсолютное "зануление" не дает возможности указать оператором LIST на какой-либо конкретный шаг в программе. При подаче же команды LIST 0, первая же строка программы с помощью использованных там управляющих кодов сделает одинаковым цвет INK и PAPER, либо закроет текст программы от чтения другим способом.

Мною разработано два метода "зануления". Первый выполняется вручную, с непосредственным Вашим участием. Второй - делает все автоматически, используя специально созданную для этих целей программу. Предпочтительно, на мой взгляд, использовать первый метод, поскольку это позволяет Вам лучше узнать структуру Бейсика и быстрее освоить принципы работы компьютера. Но, тем не менее, в этой статье мною будут рассмотрены оба подхода.

Для того, чтобы правильно произвести "зануление" вручную, необходимо четко представлять себе структуру Бейсик-строки, которую условно можно представить в виде:

MM NN <текст строки> ENTER, где:

MM - номер строки, описан двумя числами;

NN - длина строки, описана тоже двумя числами.

В интерпретаторе Бейсика под номер строки отводятся два знака, причем первым идет старший байт номера, а вторым - младший. (Для тех, кто интересуется, почему соблюдается именно такая последовательность, рекомендую прочитать подробную информацию об этом в трехтомнике "ИНФОРКОМа" Первые шаги в машинных кодах).

Текст строки совпадает с Вашим исходным текстом, за исключением того, что числа представлены в несколько ином виде. (Более подробно об этой и иных системах представления чисел читайте в том же трехтомнике в т. 1 на с. 56). Завершает строку условный код оператора ENTER - 13(тринадцать).

Для того, чтобы сделать номера строк Вашей программы нулевыми, необходимо просмотреть каждую ячейку памяти Бейсика и, зная структуру Бейсик-строки, определить, в каких адресах памяти находится код номера строки, записать номера этих ячеек, после чего с клавиатуры заслать в эти ячейки 0 командой POKE.

Для получения дампинга всех ячеек памяти Бейсика, необходимо использовать небольшую программу, которая размещается в самом конце Бейсика:

```
9997 FOR I=23755 TO 65000
9998 PRINT I; TAB 7; PEEK I; TAB 11; CHR$(PEEK I)
9999 NEXT I
```

Теперь запустим эту программу командой GO TO 9997, после чего на экране будет печататься дампинг Вашей программы (предлагаемые мною строки должны быть набраны после того, как в памяти уже находится Ваша программа) в следующем формате:

- номер ячейки памяти;
- десятичное значение числа, содержащегося там;
- символ, соответствующий данному числу.

В колонке символов, соответствующих содержимому ячеек, Вы увидите приблизительный листинг своей программы. Вашей задачей является обнаружить окончание данной строки Бейсика. Свидетельством этого служит код ENTER - 13. Причем, если десятичное значение 13 распечатается на экране, то символ CHR$ 13 является

непечатным для ZX SPECTRUM и компьютер осуществит переход на следующую строку. Обнаружив, таким образом, номер ячейки, в которой содержится код 13, свидетельствующий об окончании строки Бейсика, Вы записываете номера следующих за ним ячеек памяти, чтобы потом заслать в них 0 командой POKE.

Следует отметить, что Вам необходимо строго следить за тем, чтобы не превратить в 0 номера строк, осуществляющих "дампинг ячеек памяти". После того, как Вы завершите "зануление", Вам необходимо уничтожить строки 9997...9999.

Необходимо подчеркнуть одну маленькую деталь, помогающую облегчить работу с программой распечатки содержимого ячеек. Если эта программа по какой либо причине остановилась с сообщением об ошибке типа INVALID COLOR; NUMBER TOO BIG и т.д., то Вам достаточно набрать с клавиатуры NEXT I и "дампинг" продолжится.

Второй метод заключается в использовании программы для "зануления". Тот процесс, который Вы делали вручную, т.е. анализ строки и "зануление" ее номера, эта программа делает самостоятельно:

```
9990 REM (C) программист Михайленко Вадим Минск, МРТИ, гр 010207 Беларусь, 1991
9991 FOR i=23758 TO 65000
9992 IF PEEK i=13 THEN IF PEEK (i+1)=39 AND PEEK (i+2)=6 THEN STOP
9993 IF PEEK i=13 THEN POKE(i+1),0: POKE (i+2),0: LET i=i+4
9994 NEXT i
```

Эта программа также размещается в конце Бейсик-файла (здесь необходимо строгое соблюдение номеров строк) и осуществляет перенумерацию всех стоящих перед ней строк в 0. После работы эту программу также необходимо уничтожить. Действует же она по следующему принципу:

- в цикле идет анализ Вашей исходной программы, причем при обнаружении конца строки (об этом свидетельствует код символа ENTER - CHR$(13)), номер следующей за ней строки зануляется.

Строка 9992 осуществляет контроль таким образом, чтобы не допустить превращения в 0 (ноль) номеров строк зануляющей программы. (Они находятся по номеру 9990, поэтому наличие этой строки - обязательно). После завершения работы программа останавливается оператором STOP.

Эта программа, однако, не сможет обратить в ноль самую первую строку Вашей исходной программы. Для того, чтобы и ее обратить в ноль, Вам придется подать команду с клавиатуры:

```
POKE 23755,0
POKE 23756,0
```

Теперь Ваша программа защищена: ни одну из ее строк невозможно вызвать для редактирования.

2.3. Использование управляющих кодов.

Одной из малоизвестных и малоисследованных возможностей компьютера является использование в ZX SPECTRUM управляющих кодов. В этом разделе, включающем в себе несколько статей, я опишу применение их для защиты, для использования в Бейсик программах с целью экономии оперативной памяти, а также применение их для создания оригинального хэдера.

Но для начала я постараюсь ввести читателя в курс дела, более подробно ознакомить его с накопленным опытом использования управляющих символов.

* * *

ИНФОРКОМ рекомендует Вам также обратиться к разделу "Маленькие хитрости" в "ZX-РЕВЮ-91", где на с. 116 и 140-142 достаточно подробно разбирался вопрос использования управляющих кодов в операторе PRINT.

* * *

2.3.1. Введение

Не полностью изучив возможности своих персональных компьютеров, многие

пользователи бывают шокированы, когда остановив загрузку, чтобы посмотреть содержание фирменной программы, вместо первой строки Бейсика они видят сплошную строку (без номера) с какой-либо надписью, например:

```
CRACKED BY BILL GILBERT () год
```

Здесь может присутствовать также название программы или надписи иного рода (BILL GILBERT взят для примера - разумеется, другие "хэккеры" тоже используют управляющие коды). Текст же самой программы увидеть не удается.

С помощью управляющих кодов можно смещать текст программы в любом направлении в строке, располагать его в любом знакоместе экрана, управлять цветом символов и фона, регулировать мерцание и создавать повышенную яркость, включать инверсный режим и даже накладывать один символ на другой.

Кроме этого, специально для печати сообщений существует код, позволяющий пропускать сразу 16 символов в строке, т.е. он один как бы заменяет 16 символов типа "пробел".

Знание принципов работы управляющих кодов позволит Вам создать строку, аналогичную описанной в начале, а также так устанавливать цвета, чтобы текста программы не было видно. Кроме этого, управляющие коды можно использовать в "хэдерах" - заголовках файлов - а это позволяет стереть слово PROGRAM при загрузке, либо же расположить название программы в любой точке экрана, либо сделать и то и другое одновременно.

К таким приемам в работе с файлами прибегали многие взломщики фирменных программ (см. EXOLON, GAMEOVER и т.д.).

2.3.2. Самый первый шаг

Рассмотрим формирование строки без видимого номера и каких-либо побочных бейсиковских надписей. Суть этого процесса заключается в том, что с помощью управляющих символов мы смещаем желаемую надпись влево таким образом, чтобы она перекрывала как номер строки, так и бейсиковский оператор (обычно в подобных случаях используется оператор REM).

Изучим конкретный пример. Мы хотим, чтобы после остановки программы и подачи команды LIST у нас сверху появлялась надпись (без номера, указывающего, что это строка Бейсик-программы и без каких-либо операторов):

```
GOOD LUCK TO YOU YOUNG CRACK,
```

что в переводе с английского означает: УДАЧИ ТЕБЕ ЮНЫЙ ВЗЛОМЩИК. Для этого наберем Бейсик-строку:

```
1 REM LLLLLLLLLLGOOD LUCK TO YOU YOUNG CRACK
```

Между оператором REM и десятью символами L (символ L может быть заменен любым другим, принципиального значения это не имеет), а также между L и основной надписью Вы не должны делать пробелов. Сейчас мы имеем "полуфабрикат", который будем изменять в соответствии с ранее поставленной целью, для этого четко определим последовательность наших действий:

1) Сначала сместим особым методом исходную надпись на шесть знакомест влево.

2) Зададим цвет INK исходной надписи.

3) Зададим цвет PAPER для исходной надписи.

Будем последовательно выполнять эти пункты:

1. Для того, чтобы сместить надпись на 6 знакомест влево, Вам необходимо заслать управляющие символы оператора "курсор влево" вместо 6 символов L. Для этого наберем с клавиатуры

```
FOR i=23760 TO 23765: POKE i,8: NEXT i
```

Теперь, если Вы без ошибок набрали исходную строку с оператором REM и правильно дали команду с клавиатуры, то при подаче команды LIST Вы должны увидеть на экране надпись, содержащую 4 символа L и основную надпись, но уже без номера строки и без оператора REM.

Выполним второй пункт:

2. Возможно, что защищая свою программу, Вы пожелаете изменить цвета PAPER и INK, но нам необходимо, чтобы в любых условиях Ваша надпись была видна.

Для этих целей необходимо задать цвета непосредственно для надписи. Делается это с помощью прямых команд, засылающих вместо символов L коды управления цветом:

```
POKE 23766,16
POKE 23767,0
```

В ячейку 23766 мы засылаем код управления цветом INK, а в ячейку 23767 непосредственно значение цвета. В данном случае символы будут иметь черный цвет INK.

После этой операции на экране должны быть видны два символа L и следующая за ними основная надпись.

3. Изменим цвет PAPER. Это производится точно таким же способом:

```
POKE 23768,17
POKE 23769,7
```

В данном случае в ячейку с адресом 23768 мы занесли код управления цветом PAPER, а в ячейку 23769 непосредственно числовое значение цвета фона. Теперь надпись, появившаяся на экране после подачи команды LIST, будет содержать только то, что мы желали показать.

А если мы теперь подадим прямые команды

```
INK 0: PAPER 0: BORDER 0: CLS
```

то после подачи команды LIST получим картину, аналогичную той, которую можно наблюдать после остановки многих фирменных программ.

Теперь давайте проанализируем, что же у нас получилось из бывшей изначально достаточно простой Бейсик-строки.

23755	? 0	в этих ячейках
23756	? 1	распол. ном. строки
23757	? 44	в этих ячейках
23758	? 0	распол. длина строки
23759	REM	234 здесь наход. код оператора REM
23760	8	здесь расположен упр.
23761	8	код символа КУРСОР
23762	8	ВЛЕВО
23763	8	
23764	8	
23765	8	
23766	16	здесь находятся упр.
23767	0	коды управления цветом INK.
23768	17	здесь расположены
23769	7	упр. коды, ответственные за цвет PAPER
23770		здесь расположен наш
. . .		исходный текст: GOOD
23797		LUCK TO YOU YOUNG CRACK

Следует предупредить читателя, что для того, чтобы выполнить все вышеизложенное необходимо, чтобы системная переменная PROG, расположенная по адресу 23635-23636 указывала начало Бейсик-строки с адреса 23755. Это будет всегда, если Вы не изменяли его (следует отметить, что это значение может изменяться само при подключении некоторой переферии, в частности интерфейсов внешних устройств, например ZX-INTERFACE-1).

Рассмотрим в качестве следующего примера еще один, достаточно часто используемый хэккерами прием. Например, для того, чтобы сделать дальнейший текст программы (расположенный после строки, которую мы оформили в предыдущем примере) нечитаемым, необходимо сделать так, чтобы компьютер после подачи команды LIST сам останавливал распечатку в необходимом Вам месте. Это можно сделать несколькими способами. В качестве примера рассмотрим один из них.

После того, как компьютер распечатает строку с приветствием для взломщиков, он

должен получить команду (опять-таки с помощью управляющих символов) о том, что задаваемый нами цвет - неправильный. Если такая информация будет получена, то распечатка остановится с выдачей сообщения INVALID COLOR (неправильный цвет).

Для остановки распечатки в необходимом месте, нам понадобится зарезервировать еще 2 символа в том "полуфабрикате", который мы использовали в примере.

Следовательно, строка Бейсик-программы в первоначальном варианте будет иметь вид:

```
1 REM L...L исходный текст LL
     -----
       |
      10 символов
```

Для того, чтобы остановить распечатку, необходимо заменить пару символов LL, расположенных после исходной надписи, на управляющие коды, подобрав их значения таким образом, чтобы они создавали заведомо несуществующую комбинацию цветов.

Чтобы узнать адрес ячейки, в которой расположен последний символ основной надписи, нам необходимо к последнему значению адреса перед этой надписью (в данном случае 25769) прибавить количество символов, содержащихся в данной надписи. В надписи GOOD LUCK TO YOU YOUNG CRACK содержится 28 символов (мы должны считать не только буквы, но и пробелы). Следовательно, теперь мы можем установить адрес ячейки, в которой находится код первой буквы L, первой из тех двух, которыми мы дополнили нашу надпись:

23769+28=23797

Адрес 23797 указывает на последний символ нашей надписи, следовательно, если Вы не делали между основной надписью и парой символов LL пробел. Ячейка, содержащая первое L должна иметь следующий номер:

23797+1=23798

Проверим это, подав команду PRINT CHR$(PEEK 23798).

На экране должен появиться символ L, а для того, чтобы убедиться что мы нашли адрес ячейки именно первого L, а не второго, подадим команду:

```
PRINT CHR$(PEEK 23799)
```

На экране должно тоже появиться L. Теперь нам нужно вместо первого символа L заслать код управления цветом INK:

```
POKE 23796,16
```

После этого компьютер, анализируя данные, поступающие за управляющий символом, теоретически должен будет выполнить следующую команду:

```
INK 76
```

(т.к. код символа L - 76), но это невозможно и компьютер остановит распечатку, выдав сообщение о неправильном цвете:

```
INVALID COLOR
```

Таблица 1

	0	1	2	3	4	5	6	7	8	9
0					TRUE VIDEO	INV VIDEO	PRINT запятая	EDIT	Курсор <--	Курсор -->
10	курсор вниз	курсор вверх	DELETE	ENTER	ЧИСЛО	режим GRAPH	Упр. INK	Упр. PAPER	Упр. FLASH	Упр. BRIGHT
20	Упр. INVERSE	Упр. OVER	Упр. AT	Упр. TAB						

Описание управляющих кодов Спектрума

Таблица 2

Код	Наименование	Назначение
8	BACK SPACE	Код, создаваемый программой ввода с клавиатуры при нажатии "CAPS SHIFT" и "5". Воспринимается большинством принтеров, т.к. это ASCII код перехода назад.
9	RIGHT SPACE	Код, генерируемый программой ввода с клавиатуры при нажатии "CAPS SHIFT" и "8". Код перемещения курсора вправо, но при его использовании позиция печати не меняется. ASCII код табулирования печати строки.
10	DAWN SPACE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код заполнения строки.
11	UPSPACE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код для смещения печатной позиции вверх.
12	DELETE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код заполнения формата.
13	ENTER	Генерируется при нажатии клавиши "ENTER", выполняет возврат каретки и заканчивает печать строки при выводе на принтер. Это ASCII код перевода каретки. Предшествует номеру строки в программе.
14		Используется для чисел в 5-ти байтной форме. Это ASCII код "SHIFT OUT".
15		В стандартном БЕЙСИКе не используется. Это ASCII код "SHIFT IN". Может использоваться в расширениях БЕЙСИКа, например в БЕТА-БЕЙСИКе 3.0.
16	INK CONTROL	Управляющий код INK. Используется перед кодом, содержащим численное значение "INK". Следует отметить, что в качестве кода численного значения цвета используется не ASCII код 2, а значение 2.Например, для изменения цвета печати на экране всех символов на красный, Вам нужно использовать процедуру RST 16 с 16 в регистре A и двойкой.
17	PAPER CONTROL	Управляющий код PAPER. Используется аналогично коду INK. Значения, следующие за ним, соответствуют стандартным значениям цветов "СПЕКТРУМА".
18	FLASH CONTROL	Как и вышеизложенное, но для FLASH. Код может быть соответственно только 0 или 1.
19	BRIGHT CONTROL	Аналогично вышеприведенному для "FLASH", только в данном случае используется для "BRIGHT".
20	INVERSE CONTROL	Как и вышеприведенное, но для "INVERSE".
21	OVER CONTROL	Как и вышеприведенное, но для "OVER".
22	AT CONTROL	Код контроля "AT", который должен сопровождаться номером строки и столбца соответственно.
23	TAB CONTROL	Как и вышеприведенное, но для "TAB". В данном случае код должен сопровождаться только значением столбца.

Если Вы заблаговременно "занулили" строки своей основной программы а также "занулили" первую ее строку, т.е. ту, которая формирует данную надпись, подав команды

```
POKE 23756,0:
POKE 23755,0
```

то теперь, подав команду LIST Вы получите на экране сообщение, которое составляло нашу исходную надпись, а продолжить распечатку никак не сможете т.е. невозможно указать на последующие строки Вашей программы с помощью оператора LIST - все они имеют

нулевой номер.

Команда же LIST 0 снова распечатает строку с исходным сообщением.

В примере 2 для того, чтобы скрыть дальнейший листинг, мы создали заведомо неправильный цвет, но можно поступить иначе, например, сделать после исходной надписи одинаковыми цвета INK и PAPER. Это обеспечит не меньшую надежность в сокрытии листинга. Подадим команды:

```
POKE 23798,16
POKE 23799, 7
```

И, в тоже время, Ваша защита будет иметь серьезный вид, если Вы вообще уберете с экрана какие-либо надписи, т.е. после подачи команды LIST экран будет продолжать оставаться чистым. Это может быть достигнуто в том случае, если мы сделаем одинаковыми цвета символов и фона еще до исходной надписи (строго говоря, необходимость надписи в этом случае отпадает).

Это делается после смещения строки на 6 пикселей влево (см. пример 1 пункт 1) подачей команд:

```
POKE 23766,16
POKE 23767,7
POKE 23768,17
POKE 23769,7
```

На этом наш беглый обзор применения управляющих кодов можно закончить, но в последующих статьях материал будет рассмотрен более детально. Вместе с тем, чтобы Вы начинали не с нуля, рекомендуется проделать описанные эксперименты на компьютере. Это будет способствовать лучшему пониманию материала.

2.3.3. Управляющие коды ZX SPECTRUM.

Управляющие коды в ZX SPECTRUM используются так же, как и в других компьютерах для управления печатью на экране и на принтере. Ниже приведены таблицы 1 и 2.

Внимательно изучив предлагаемый материал, Вы сможете полноценно использовать управляющие коды в своих разработках.

(ПРОДОЛЖЕНИЕ СЛЕДУЕТ)

Дж. Хардман, Э.Хьюзон Перевод Тихоновой Н. А.

40 ЛУЧШИХ ПРОЦЕДУР

Данная книга является сокращенным переводом книги "40 Best Machine Code Routines For The ZX Spectrum With Explanatory Text", J.Hardman & A. Hewson, содержащей в себе набор программ в машинных кодах с весьма подробными разъяснениями принципов их работы.

Значительным сокращениям подверглись те разделы оригинала, в которых рассматриваются вопросы компьютерной терминологии и система команд Z80 - советуем обратиться к трехтомнику по программированию в машинных кодах, выпущенному "Инфоркомом".

При подготовке данного пособия были также исключены описания программ, которые не представляют, на наш взгляд, особого интереса.

РАЗДЕЛ А

1. ВВЕДЕНИЕ

Цель этой книги - обеспечить как начинающего, так и опытного пользователя компьютера ZX SPECTRUM полезными, интересными и развлекательными программами в машинных кодах. Книга имеет 2 раздела.

Раздел А описывает те особенности SPECTRUMа, которые важны при программировании в машинных кодах, некоторые процедуры системного ПЗУ, а также структуру машинного языка.

Раздел В представляет сами программы. Они представлены в стандартном формате, который детально описан в начале раздела. Программы полностью закончены, т.е. они могут быть загружены и использованы индивидуально, без обращений к другим программам.

Предлагаемые программы могут быть загружены с помощью простого загрузчика машинных кодов (MC-LOADER - маш. ЗАГРУЗЧИК), описанного в начале раздела В.

Общие сведения о Бейсике и машинных кодах

Микропроцессор Z80A, на базе которого сделан ZX SPECTRUM, не понимает непосредственно слова БЕЙСИКа. Такие, как PRINT, IF, TAB, и т. д. Вместо этого он выполняет приказы специального языка - своего внутреннего машинного кода. Процедуры ПЗУ, которые придают SPECTRUMу его индивидуальность, написаны на этом специальном языке и состоят из большого количества стандартных подпрограмм для ввода-вывода листинга, интерпретирования и выполнения команд BASICа и др.

Например, стандартные подпрограммы говорят процессору Z80A ("ЧТО ДЕЛАТЬ, ЕСЛИ..."). Если, например, команда BASICа - слово PRINT, то что делать, если следующий элемент имя переменной; или что делать, если следующий элемент - запятая и т. д.

Машинный код состоит из последовательности положительных целых чисел (от 0 до 255), которые диктуют действия для Z80A. Хотя машина использует двоичную форму представления чисел, нет необходимости для человека изучать команды в такой форме. Мы будем использовать десятичную форму, которая обрабатывается МС - ЗАГРУЗЧИКОМ из Раздела В.

Однако даже однообразную строку десятичных чисел трудно интерпретировать и поэтому десятичные числа обычно преобразовываются в специальный язык (ассемблер), который представляет собой определенные аббревиатуры. Язык ассемблера называется так потому, что специальная программа, называемая АССЕМБЛЕРОМ, используется для обработки ("сбора или ассемблирования") команд в машинных кодах при написании (формировании) программы.

Требуется только одно число, чтобы точно определить простую команду Z80A. Например, команда СКОПИРОВАТЬ содержимое регистра С в регистр D - это десятичное число 81 (термин "регистр" более детально описан в главе 3, а пока достаточно если Вы

будете воспринимать C и D, как переменные BASICа). Для таких команд есть точное соответствие между десятичным числом и командой. 81, например, записывается на языке АССЕМБЛЕРа, как LD D,C ("LD", кстати, сокращение слова "load" загрузить). Многие команды ассемблера состоят из подобных простых аббревиатур по этой причине они часто называются мнемониками. Более сложные команды требуют 2, 3 или 4 числа. Но, все равно, для представления их используется одна команда ассемблера. Табл. 1.1. показывает список нескольких чисел, их мнемоник и краткое объяснение действия Z80A.

Таблица 1.1. Некоторые примеры машинных команд Z80A

Ссылка	Десятичное число	Мнемоника	Комментарий
(a)	81	LD D,C	Загрузить в D содержимое C
(b)	14 27	LD C,27	Поместить число 27 в C.
(c)	14 13	LD C,13	Поместить число 13 в C.
(d)	33 27 52	LD HL,13339	Поместить 13339 в пару регистров HL. Обратите внимание: 27+256*52= 13339; 27 поместить в L; 52 поместить в H.
(e)	221 33 27 52	LD IX,13339	Поместить 13339 в пару регистров IX.

Строка (a) таблицы - пример LD D,C рассматривался выше, строки (b) и (c) показывают, как положительное число может быть загружено в регистр (используются два числа: первое определяет действие, которое должно быть выполнено, а второе определяет число, которое должно быть загружено). Строка (d) показывает, как большое целое число может быть загружено в два регистра (H и L) вместе. Здесь второе и третье числа определяют, какие числа должны быть загружены. Последний пример в строке (e) иллюстрирует четырехбайтовый код для загрузки большого целого числа в пару регистров IX. Обратите внимание, что три из четырех чисел такие же, как в строке (d). А дополнительное первое число определяет пару регистров IX вместо HL.

Структура машинного языка объясняется более подробно в главе 3, а полный список мнемоник ассемблера Z80A можно найти в литературе по микропроцессорной технике и программированию.

Итак, наиболее насущный вопрос:

ЗАЧЕМ ИСПОЛЬЗУЮТ МАШИННЫЙ КОД?

На некоторых компьютерах это делается потому, что задачи, которые пользователь желает выполнить, слишком медленно выполняются. В этом отношении ZX SPECTRUM не исключение. Рассмотрим, например, проблему сохранения полного отображения экрана в RAM (ОЗУ) или копирования его обратно на экран с целью создания эффекта мультипликации.

Файл изображения и цветовые атрибуты занимают 6912 байт. Следующая программа BASICа сохранит отображение экрана, но это займет много времени - около 70 секунд:

```
5 CLEAR 58623
10 FOR I=0 TO 6911
20 POKE 58624+I,PEEK (16384+I)
30 NEXT I
```

Причина такой медленной работы в том, что SPECTRUM тратит больше всего времени на декодирование команд BASICа перед их выполнением. Некоторое количество времени также тратится на преобразование чисел в двухбайтную форму (которую понимает Z80A) из десятичных чисел в пятибайтной форме (с которыми оперирует BASIC), а также на выполнение пятибайтовой арифметики.

Так, в нашем примере по переброске экрана должны быть выполнены следующие шаги:

1. Прибавить i к 16384.
2. Преобразовать результат в форму двух байтов.
3. Восстановить содержимое адреса PEEK.

4. Прибавить i к 58624.
5. Преобразовать результат в форму 2-х байтов.
6. Сохранить полученное (PEEK) значение по заданному адресу (POKE).
7. Прибавить 1 к значению I и сохранить результат.
8. Вычесть i из 6911. Если результат положительный, то идти на пункт 1.

Во время прохождения цикла, SPECTRUM должен декодировать каждую команду снова, так как в данном случае память не используется для сохранения последовательности предыдущих действий. Легко увидеть, что компьютер тратит более 99 процентов времени на подготовку к выполнению задачи, а не на выполнение самой задачи. Аналогичная программа в машинных кодах для сохранения экрана выполняется практически мгновенно. Пример такой программы дан в Разделе B.

2. ВНУТРЕННЯЯ СТРУКТУРА ZX SPECTRUM

Компьютер - это машина, которая способна запомнить последовательность команд и затем выполнить их. Конечно, чтобы сделать так, требуется память, в которой команды могут быть сохранены. ZX SPECTRUM имеет два типа памяти. Первый тип - ПЗУ (ROM), в котором содержится фиксированная последовательность инструкций, введенных в машину ее изготовителем.

Второй тип - ОЗУ (RAM). RAM - это "блокнот для записей" компьютера. Когда компьютер выполняет задачу, он непрерывно просматривает, что находится в RAM ("чтение" из памяти) и обновляет содержимое RAM ("запись" в память). SPECTRUM не использует свой "блокнот" для записей случайно. Различные части RAM используются для хранения различных видов информации. Программа BASICa, введенная пользователем, например, хранится в одной части RAM, в то время как переменные, используемые этой программой, хранятся в другом месте. Размер "блокнота" для записей ограничен, и потому машина точна при распределении пространства для информации, которую она хранит. Свободное пространство собрано в одном месте, и, если пользователь хочет добавить строку в свою программу, информация в RAM должна быть "перетасована" по всей длине, используя некоторую свободную область для вставки дополнительной строки.

В значительной степени эта глава посвящена объяснению организации RAM SPECTRUMa, так как многие программы раздела B предназначены для манипуляций с RAM. Глава содержит в себе описание дисплейного файла, атрибутов, буфера принтера, системных переменных, программной области и области программных переменных. В конце раздела описываются стандартные подпрограммы из ROM, к которым ссылаются программы в Разделе B.

Карта памяти

RAM имеет 49152 ячейки памяти. Каждая ячейка может хранить одиночное целое число от 0 до 255 включительно и задается адресом, который является положительным целым числом от 0 до 65535. Адреса от 0 до 16383 зафиксированы для постоянной памяти - ROM. Первый адрес RAM (ОЗУ) - 16384. Табл. 2. 1. - упрощенная карта памяти SPECTRUMa, которая показывает, как используется RAM с адреса 16384.

Дисплейный файл, который хранит отображаемую на экране информацию, занимает ячейки от 16384 до 22527. Атрибуты, которые определяют цвет, яркость и т.д. для каждого знакоместа экрана следуют непосредственно дальше: в ячейках 22528 - 23295.

Первые 5 адресов в колонке Таблицы 2.1. являются фиксированными, т.к. дисплейный файл, атрибуты и т.д. занимают фиксированное положение. Пятая область назначена для карты памяти микродрайва. Это небольшое периферийное устройство представляет собой нечто среднее между магнитофоном и дисководом. Грубо говоря - это дешевая альтернатива дисководу. Скорость его работы достигаете за счет того, что, с одной стороны, лента движется с очень высокой скоростью, а с другой - за счет организации прямого доступ к файлам, как на диске, а не последовательного, как на магнитофонной кассете. Вот для того чтобы иметь в некоем месте хранилище с данными о том, что записано на каждом секторе микродрайва в памяти компьютера и выделен область карты памяти

микродрайва. Если микродрайв (а точнее INTERFACE-1, через который он подключается), подсоединен к SPECTRUMу, эта область содержит информацию о его секторах. Если же не подсоединен, эта область не нужна и в этом случае шестая область (информация о каналах) размещена непосредственно за четвертой областью, системными переменными, т.е. стартовый адрес области информации о каналах и всех последующих областей не фиксирован, а может "плавать" вверх-вниз в RAM. SPECTRUM хранит стартовый адрес всех этих областей в системных переменных. Область системных переменных находится перед картой микродрайва в ячейках 23552-23733 включительно. Эти адреса являются все время строго фиксированными.

Стартовый адрес или имя системной переменной	Ячейка системной переменной	Содержимое памяти
16384	-	Дисплейный файл
22528	-	Атрибуты
23296	-	Буфер принтера
23552	-	Системные переменные
23734	-	Карта микродрайва
CHANS	23631	Область информации о каналах
PROG	23635	Адрес начала программы на БЕЙСИКе
VARS	23627	Адрес начала области программных переменных.
E-LINE	23641	Адрес области редактирования
WORKSP	23649	Буфер INPUT
STKBOT	23651	Стек калькулятора
STKEND	23653	Свободная область
SP	-	Машинный стек и стек GO SUB
RAMTOP	23730	Пользовательские подпрограммы в машинном коде.
UDG	23675	Графика пользователя.
P-AMT	23732	Физическая вершина ОЗУ.

Таблица 2.1. Карта памяти. Указатель стека SP хранится не в RAM, а в SP -регистре микропроцессора Z80A.

Адреса ячеек, которые хранят стартовые адреса всех "плавающих" областей, даны в колонке 2 таблицы 2.1. Адрес области начала программы BASICа, например, хранится в ячейках 23635 и 23636 в области системных переменных.

Примечание: Ссылка к системной переменной с помощью адреса, по которому она хранится, довольно неудобна. По этой причине в Таблице 2.1. в 1-ой колонке даны условные имена системных переменных. Эти имена удобны только для пользователей, в то время как SPECTRUMОМ они, естественно, не распознаются. Например, введенная строка:

```
PRINT PROG
```

даст сообщение об ошибке

"2: variable not found" ("Переменная не найдена").

PEEK и POKE

Карта памяти - это ключ для понимания того, как компьютером используется RAM-память. Для непосредственного управления RAM используются ключевые слова BASICа - PEEK и POKE, которые позволяют просмотреть и изменить содержимое любой ячейки памяти. PEEK - функция вида:

```
PEEK <адрес>
```

Адрес - это целое положительное число от 0 до 65535 или арифметическое выражение, которое, выполняясь, дает положительное число. Важно заключить арифметическое выражение в скобки, т.к. PEEK 16384 + 2 интерпретируется, как (2 + результат PEEK 16384), тогда как PEEK (16384 + 2) интерпретируется, как PEEK 16386

Значение, выдаваемое функцией PEEK - это число, хранящееся по данному адресу в текущий момент. Оно всегда будет положительным от 0 до 255.

Выше объяснялось, что системная переменная PROG хранится по адресу 23635, но значение PROG (т.е. адрес в RAM) всегда больше числа 255. Следовательно, для его хранения необходимы две смежных ячейки с адресами 23635 и 23636. Значение PROG может быть получено с помощью командной строки:

```
PRINT "PROG="; PEEK 23635 + 256 * PEEK 23636
```

Все адреса хранятся в 2-х смежных ячейках в такой форме и могут быть получены вводом: PRINT PEEK 1-я ячейка + 256 * PEEK 2-я ячейка

Например, если SPECTRUM используется без подсоединенного микродрайва, область карты микродрайва не будет существовать, и информация о каналах будет располагаться непосредственно после области системных переменных, т.о. системная переменная CHANS должна быть такой же, как стартовый адрес карты микродрайва, когда он существует, т.е. 23734. CHANS хранится в 23631 и 23632 и, следовательно, после ввода: PRINT PEEK 23631 + 256 * PEEK 23632 будет получено значение 23734.

Функция PEEK может быть использована также для просмотра содержимого любой из ячеек ПЗУ. Это очень полезно. Просмотр любой ячейки не приводит к разрушению или искажению программы или переменных. Иногда результаты PEEK могут быть обманчивыми, т.к. содержимое ячейки, которая просматривалась, может изменяться в течение или непосредственно после выполнения команды просмотра.

Например, если просматривается содержимое ячеек, которые связаны с левым верхним углом экрана дисплея, то результаты, распечатанные в верхнем левом углу экрана, будут уже устаревшими, т.к. в момент вывода изображения мы уже изменили эти ячейки.

Команда POKE более рискованная, чем функция PEEK, т.к. задавая ее, пользователь вмешивается в функционирование компьютера. Использование этой команды может быть причиной сбоя машины или ее останова.

Формат команды:

POKE <адрес>,<число>

Адрес - это положительное целое число от 0 до 65535 включительно или арифметическое выражение, которое дает такое число. В этом случае нет необходимости заключать арифметическое выражение в скобки, т.к. POKE - это команда, а не функция (хотя есть негласная заповедь, что лишними скобками программу не испортишь). Число, помещаемое в ячейку, может быть от 0 до 255 включительно.

Дисплейный файл

Обычно дисплей содержит 24 строки по 32 символа. Дисплейный файл занимает ячейки от 16384 до 22527, т.е. 6144 ячейки в общей сложности. Следовательно, количество ячеек, используемое для символа:

6144/(24*32)=8.

Эти 8 ячеек формируют изображение символа экране, называемое знакоместом.

Наиболее легкий путь получения общего впечатления о том, как организован дисплейный файл - это печать (PRINT) картинки на экране, сброс (SAVE) экрана на ленту, очистка экрана (CLS) и загрузка (LOAD) картинки экрана вновь. Программа P2.1 сохраняет (SAVE) и загружает (LOAD) экран, используя графический символ на клавише 5 для создания оригинальной картинки.

Программа P2.1.

```
100 FOR  I=0 TO 703
110 PRINT " "; : REM символ, находящийся на клавише "5" в G-режиме
120 NEXT I
130 SAVE  "Picture"  SCREEN$
140 CLS
150 INPUT "Перемотайте ленту и включите воспроизведение"; z$
160 LOAD  "Picture"  SCREEN$
```

Когда картинка загружается с ленты, видно, что дисплей разделен на три зоны по 8 символьных строк в каждой, а каждая строка разделяется на восемь пиксельных линий. SPECTRUM загружает сначала верхние пиксельные линии для первых восьми строк, затем следующие пиксельные линии тех же восьми строк и т.д. Таким же образом формируются средняя и нижняя части дисплея.

Другой путь понимания формирования дисплея - это рассмотреть, где хранятся 8 байтов, которые используются для формирования символа в верхнем левом углу экрана. Первый байт формирует самую верхнюю 1/8 часть символа и размещается в начале дисплейного файла по адресу 16384.

Команда POKE 16364,0 очистит верхнюю линию пикселей самого верхнего левого знакоместа, в то время, как POKE 16384,255 закрасит всю эту линию. При помещении в эту ячейку числа от 0 до 255 мы получим в этом месте экрана различные штрихи. Вторая сверху линия в первом знакоместе на экране не сформирована числом, хранящимся в ячейке 16385, - эта ячейка используется для верхней линии пикселей в соседнем символе. Вторая линия сверху в первом знакоместе формируется числом, хранящимся в ячейке 16384+32*8=16640.

Подобным же образом вычисляются адреса оставшихся шести линий этого знакоместа.

Следовательно, образ символа в верхнем левом углу экрана определяется содержимым адресов 16384, 16640, 16896, 17152, 17408, 17664, 17920, 18176.

Программа P2.2. позволяет Вам экспериментировать, помещая различные числа в эти восемь ячеек.

Программа P2.2. Программа для создания символа в верхнем левом углу экрана.

```
10 REM подпрограмма установки символа в верхнем углу экрана
20 INPUT "Символ состоит из восьми байтов, каждый из которых - число в диапазоне от 0 до
     255. Введите номер байта (от 0 до 7)";n
30 IF n<0 OR  n>7 OR  n<>INT n THEN BEEP  0.2,24: GO TO 20
40 INPUT "Ввести содержимое байта";m
50 IF m<0 OR m>255 OR m<>INT m THEN BEEP  0.2,24: GO TO 40
60 POKE 16384+8*32*n,m
```

Программа P2.3. Программа декодирования атрибута.

```
10 REM декодер атрибутов
20 DATA "Black", "Blue", "Red", "Magenta", "Green", "Cyan", "Yellow", "White", "Bright",
     "Flash"
30 DIM C$(8,7)
40 FOR  I=1 TO 8
50 READ  C$(I)
60 NEXT I
100 REM Декодер атрибутов
110 INPUT "Ввести число от 0 до 255. Эта программа декодирования интерпретирует его в файл
     атрибутов";n
120 IF  n<0 OR  n>255 OR  n<>INT n THEN BEEP  .2,24: GO TO 110
200 PRINT "Цвет символа - "; c$(1+n-8*INT(n/8))
210 PRINT "Цвет фона - "; C$(1+INT(n/8)-8*INT(n/64))
220 IF INT(n/64)=1 OR INT(n/64)=3 THEN PRINT "СИМВОЛ - BRIGHT"
230 IF n>127 THEN PRINT  "Символ будет мерцать (FLASH)"
300 PRINT AT 6,0; "::::::::::::::::::::::::::::::::"
310 FOR i=22720 TO 22751
320 POKE i,n
330 NEXT i
500 INPUT "Для повторения - ENTER";z$
510 CLS
520 GO TO 110
```

Каждая ячейка в дисплейном файле определяет восемь пикселей на экране. При этом

число, которое хранится в данной ячейке, преобразуется в двоичную форму и затем устанавливаются восемь пикселей, соответственно двоичным цифрам. Например, 240 после преобразования в двоичное число дает:

11110000

Следовательно, если ячейка содержит число 240, четыре из восьми пикселей, соответствующие единицам, будут высвечены, а оставшиеся нет. Следовательно, дисплейный файл состоит из 6144 ячеек памяти по 8 ячеек для одного знакоместа.

Каждая ячейка определяет горизонтальную полосу из восьми пикселей. Ячейки, относящиеся к данному знакоместу, не расположены последовательно одна за другой.

Атрибуты

Содержимое дисплейного файла определяет, какие пиксели высвечиваются на экране, цвет фона (PAPER), символа (INK), яркость (BRIGHT) и мерцание (FLASH) определяются с помощью атрибутов. Область атрибутов занимает ячейки 22528 - 23295 - по одной ячейке для каждого из 768 знакомест.

Соответствие между содержимым ячеек памяти файла атрибутов и самими атрибутами - следующее:

Значение атрибута = 128*FLASH + 64*BRIGHT + 8*PAPER + INK

FLASH и BRIGHT принимают значение 1, если соответствующее условие установлено, а PAPER и INK принимают значение требуемого цвета, как показано на клавиатуре (красный, например 2).

Программа P2.3. декодирует атрибуты, т.е. данное значение атрибута распечатывает с соответствующим цветом PAPER и INK.

Буфер принтера

256 ячеек ОЗУ, следующие за областью атрибутов, используются, как буфер для хранения строки символов, которая должна быть передана на принтер.

Многие программы в Разделе В будут использовать буфер принтера для передачи данных BASICа или от клавиатуры в подпрограммы. Буфер подходит для этой цели, т.к. его ячейки фиксированы и маловероятно, что пользователь пожелает его использовать для других целей.

Единственно важное ограничение в этом случае - не использовать команды BASICа, которые требуют работы с принтером - LLIST, LPRINT, COPY.

Есть еще и ограничение для владельцев 128-килобайтных машин. У них нет буфера принтера. Дело в том, что буфер принтера предназначался в оригинальной модели для поддержки недорогого специализированного узкопечатного ZX-принтера. В фирменной модели 128-килобайтных машин есть порт подключения полноценного матричного принтера, обладающего собственным буфером и необходимость в этом буфере отпала, зато в этих моделях необходимо больше системных переменных и под них отдали область буфера ZX-принтера. Теперь, если в режиме 128K пользователь что-либо зашлет в эту область, то нарушив системные переменные он выведет программу из строя.

Область программ на BASICе

Обычно эта область начинается с адреса 23755 и на нее указывает содержимое системной переменной PROG (23635,23636), но есть и исключения за счет некоторых видов периферийных устройств.

Если, например, к компьютеру подсоединен микродрайв, то начало этой области сдвигается, в этом самом общем случае начало области программ на BASICе и определяют с помощью системной переменной PROG. Ниже предполагается, что такая периферия не подсоединена.

Программа P2.4. распечатывает содержимое 18-ти ячеек в начале программной области, как показано на рис. 2.1. В этих 18 ячейках хранится первая строка данной программы.

```
10 REM Peek program
```

```
20 FOR i=23755 TO 23772
30 PRINT i,PEEK i
40 NEXT i
```

Программа P2.4. Программа для просмотра содержимого первых 18-ти ячеек в программной области.

23755	0
23756	10
23757	14
23758	0
23759	234
23760	80
23761	101
23762	101
23763	107
23764	32
23765	112
23766	114
23767	111
23768	103
23769	114
23770	97
23771	109
23772	13

Рис.2.1. Форма, в которой строка 10 REM peek program хранится в программной области.

Номер строки (10) хранится в первых двух ячейках в форме:

Номер строки = 256*PEEK первый адрес + PEEK 2-й адрес.

Следующие 2 ячейки: 23757 и 23758 хранят длину оставшейся части строки, начинающейся в ячейке 23759. В нашем случае: 14+256*0=14

Т.о. следующая строка начинается с ячейки:

23759+14=23773

Ячейка 23759 хранит в себе число 234, которое является кодом ключевого слова (токена) REM. Следующие 12 ячеек хранят коды символов одиннадцати букв и пробела, составляющих фразу Peek program. Последняя ячейка хранит число 13, которое является кодом для ENTER, определяющим конец строки

Таблица 2.2. показывает метод кодирования программы в программной области.

Ячейки	Номер строки
1 и 2	Номер строки
3 и 4	Длина строки, исключая первые 4 ячейки
5	Код команды
Конец	Символ ENTER, число 13

Табл. 2.2. Этот метод используется для кодирования строк программы.

Момент, который опущен в таблице - это описание метода хранения числовых значений, имеющих место в программе. Этот метод может быть исследован на примере строки:

```
10 LET a=1443
```

Введите ее в программу P2.4. На рис.2.2 показан результат работы программы в этом случае.

23755	0
23756	10
23757	14
23758	0
23759	241
23760	97
23761	61
23762	49
23763	52
23764	52
23765	51
23766	54
23767	0
23768	0
23769	163
23770	5
23771	0
23772	13

Рис. 2.2. Формат, в котором строка 10 LET a=1443 хранится в программной области.

Ячейки 23755-23758 такие же, как в прошлом примере. Затем следуют коды для LET, a, =, и четыре кода цифр, которые вместе формируют число 1443. Следующий элемент, находящийся в ячейке 23766, - это код 14. Этот код указывает, что следующие 5 ячеек хранят число в специальном пятибайтном формате. Линия заканчивается в ячейке 23772 вводом символа ENTER, как и ранее.

*   *   *

Примечание ИНФОРКОМа. Получается так, что в одной строке число как бы записано дважды. Первый раз своими символами, а второй раз - в пятибайтной форме после кода 14. Зачем это нужно?

Дело в том, что первое представление используется для того, чтобы БЕЙСИК-интерпретатор знал, что вам показать на экране, а второе - используется для расчетов во внутреннем калькуляторе компьютера. (О калькуляторе читайте в т. 1 и 2 нашего трехтомника).

Самое интересное, что они могут и не совпадать. В этом случае Вы на экране будете видеть одно, а программа будет обрабатывать совсем другое число, и этим нередко пользуются в защите программ. Например, Вы видите на экране LET a=1257: GO TO a и пытаетесь проследить работу программы, а на самом деле там было записано нечто совсем другое и компьютер делает переход не к строке 1257, а туда, куда надо.

Желающие могут посмотреть загрузчик программы BOMBJACK. В череде относительно несложных вывертов этот там стоит одним из первых. Но если на него "клюнуть", атака на программу никак не получится.

*   *   *

Цифровой пятибайтный формат

Пять ячеек памяти используются для хранения чисел в программе на BASICе (исключая номера строк). Целые числа в диапазоне от -65535 до +65535 хранятся таким же образом, как в формате Z80A. Для этих чисел первые две ячейки и последняя содержат 0, а третья и четвертая хранят число в двухбайтной форме:

Число = PEEK 3-я ячейка + 256* PEEK четвертая ячейка

Таким образом 16533 хранится в пяти ячейках как

```
0    0    169  64   0
```

потому, что

169+256*64=16553

Дробные числа хранятся в формате с плавающей запятой таким образом: экспонента в первой ячейке, а мантисса в следующих четырех ячейках, т.е.:

число = мантисса * 2^экспонента

Первая ячейка мантиссы используется также для определения знака числа. Если ячейка содержит значение в пределах от 0 до 127, число положительное, если нет - отрицательное.

Программа P2.5 может быть использована для восстановления дробного числа из составляющих его пяти компонентов.

```
10 PRINT "Ввести экспоненту и четыре байта мантиссы. Все числа должны находиться между 0 и
     255 включительно."
20 INPUT e,a,b,c,d
30 PRINT ,, " Exponent= ";e
40 PRINT "Mantissa=  "; a,,b,,c,,d
50 PRINT ,,"The number=  "; (2*(a<128)-1)*2^(e-160)* (((256*(a+128*(a<128))+b)*256+c)*256+d)
```

Программа P2.5. Эта программа восстанавливает дробное число.

Область переменных

Область переменных начинается в ячейке, адрес которой хранится в системной переменной VARS (ячейка 23627). Как бы ни была объявлена новая переменная, т.е. в программе или непосредственным вводом с клавиатуры, для нее резервируется соответствующее количество свободного пространства в этой области.

Все имена переменных начинаются с буквы. Различий между верхним и нижним регистром нет. Эти ограничения позволяют SPECTRUMу манипулировать с кодом первого символа каждой переменной и, таким образом, он может различить шесть типов переменных просмотром диапазона, в котором находится код.

Все цифровые переменные с односимвольными именами, например, имеют коды в пределах от 97 до 122; буква a - 97; b - 98; c - 99 и т.д. Подобным же образом цифровые массивы имеют коды в пределах 129 – 153, т.е. a - 129; b - 130; c - 131 и т.д. Диапазоны кодов представлены в таблице 2.3.

Длина каждого типа переменной также показана в таблице 2.3.

Тип переменной	Диапазон символьного кода	Длина в области переменных
Цифровой (односимвольное имя)	97 - 122	6
Цифровой (многосимвольное имя)	161 - 186	5 + длина имени
Цифровой массив	129 - 154	4 + 2 * размерность + 5 * общее количество элементов.
Управляющая переменная цикла FOR-NEXT	225 - 250	18
Символьная строка	65 - 90	3 + длина строки
Символьный массив	193 - 218	4 + 2 * размерность + общее число элементов.

Таблица 2.3. Переменные. Диапазон кодов и длина переменных.

Подпрограммы ПЗУ

Некоторые из представленных в книге программ (раздел B) используют стандартные подпрограммы ПЗУ:

RST 16 - распечатывает содержимое аккумулятора.

CALL 3976 - вставляет символ, хранящийся в аккумуляторе, по адресу, хранящемуся в паре регистров HL.

CALL 6326 - если аккумулятор хранит код 14, устанавливается нулевой флаг и

увеличивается пара регистров HL в пять раз.

CALL 6510 - возврат в HL адреса в ОЗУ той строки, номер которой был передан в эту подпрограмму через HL.

3. МАШИННЫЙ ЯЗЫК Z80

Регистры Z80A

Во время выполнения программы компьютер не обновляет непосредственно содержимое памяти. Он копирует содержимое ячейки памяти в регистр и оперирует содержимым регистра. Регистры в машинном языке имеют функцию, схожую с переменными в BASICе, т.е. используются для хранения чисел. Они отличаются от переменных BASICа тем, что число их ограничено и они существуют в процессоре, а не в RAM. Кроме того, они могут хранить только один байт, или 2 байта в паре регистров.

Z80A - имеет несколько регистров и потому может хранить несколько чисел одновременно. Благодаря этому снижается время при обмене информацией между процессором и памятью.

Регистр "A" (Аккумулятор)

Аккумулятор - это наиболее важный регистр, т.к. больше всего арифметических и логических команд выполняется с содержимым этого регистра. Этот регистр называется аккумулятором, т.к. результат последовательных операций накапливается ("аккумулируется") в нем. Некоторые команды, которые обращаются к аккумулятору, используют другой регистр или адрес памяти в качестве источника данных. Например, инструкция

```
ADD A,B
```

дает процессору команду прибавить содержимое регистра B к содержимому регистра A, поместив результат в A.

Флаг	Мнемоника		Использование
Знак	N	P	Включается, когда результат последней операции отрицательный.
Нуль	Z	NZ	Включается, когда результат последней операции равен нулю или имело место совпадение.
Перенос	C	NC	Включается, когда происходит переполнение регистра, т. е. последний результат больше того, что может быть записанным в один байт (или в два байта для операций с парой регистров).
Четность/ Переполнение	PE	PO	Флаг включается, если в байте результата предыдущей операции количество включенных битов есть величина четная. В некоторых операциях этот флаг свидетельствует о переполнении.

Таблица 3. 2. Четыре флага, которые контролируют наибольшее количество операции Z80A.

Флаг-регистр "F" (регистр состояний)

Регистр F довольно значительно отличается от всех остальных, т.к. его содержимое не рассматривают как один байт, а рассматривают как 8 индивидуальных битов, что конечно же одно и то же. Эти биты используются в качестве так называемых флагов для управления последовательностью выполнения программы. Каждый флаг используется для определения того, какое из двух логически противоположных условий имело место при выполнении предыдущей операции. Например, флаг нуля определяет, был ли равен нулю результат последней операции сложения, вычитания и т. п. Только 4 из восьми флагов наиболее интересны для пользователей, их свойства кратко изложены в Таблице 3.2.

Флаг знака наиболее простой. Кроме того, условленно, что в операциях со знаковыми числами седьмой бит (старший бит в байте) тоже используется для хранения знака. Он включается, когда число отрицательно, иначе снимается.

Этот флаг отражает знак последнего результата.

Флаг нуля устанавливается, если результат последней операции равен нулю. Он также используется инструкциями сравнения, которые фактически аналогичны вычитанию, при котором результат игнорируется.

Флаг переноса регистрирует переполнение, которое имеет место, если результат сложения длиннее чем то, что может быть записано в регистр, либо имеет место заем при вычитании. Имеется также несколько инструкций сдвига, в которых биты в регистре A сдвигаются влево или вправо циклически через флаг переноса.

Флаг четности/переполнения - это два флага в одном. Он используется, как флаг переполнения при выполнении арифметических операций для определения, был ли бит 7 получен в результате переноса или сгенерирован битом 6 при "взятии займа". Это используется в случае, если бит знака был искажен. Логические инструкции используют тот же самый флаг для определения четности результата.

Примечание: четность двоичного числа - это четность количества битов, установленных в единицу. Если количество четно, то говорят, что имеет место четность, если оно нечетно, то говорят, что имеет место нечетность. Флаг устанавливается, если имеет место четность.

Результат некоторых инструкций зависит от текущих установок отдельных флагов, например, инструкция

```
JR Z, D
```

позволяет Z80A "перепрыгнуть" через несколько инструкций, если флаг нуля установлен (т. е. сделать условный переход типа IF...THEN GO TO). Если флаг нуля не установлен, процессор выполняет следующую инструкцию в обычной последовательности, т.о. флаговый регистр очень важен, т.к. он позволяет процессору принимать решения и переходить к другой части программы.

Регистры счетчики "B" и "C"

Регистр B и, в некоторой степени, регистр C, с которым он может использоваться в паре, доступен для нескольких применений. Но наиболее важно использование его в качестве счетчика. Мы уже видели, как выполнение программы может управляться использованием флага 0 в инструкции JR z, n.

Другая инструкция: DJNZ n использует флаг 0 для создания циклов, используя регистр B в качестве счетчика аналогично циклам FOR-NEXT в BASICе. Когда встречается эта инструкция, Z80A уменьшает содержимое регистра B на 1. Если результат равен нулю, то выполняется следующая инструкция в нормальной последовательности, если результат не равен нулю, подпрограмма "перескакивает" n инструкций. Если программист использует отрицательное значение для n, "прыжок" осуществляется в программе назад и, если здесь нет других переходов, процессор, конце концов, встретит ту же самую инструкцию вновь.

Т.о. загрузкой регистра B определенным числом и установкой соответствующего смещения n определенная часть кодов может быть выполнена заданное количество раз. Регистр B содержит только один байт и поэтому может устанавливаться на любое число от 0 до 255, т.е. используя описанный механизм через определенный блок кодов может быть сделано максимум 255 проходов. Для осуществления более 255 проходов в цикле нет эквивалентных инструкций, но имеются очень мощные инструкции, которые используют все 16 битов регистровой пары BC, как счетчик с максимальным значением 65535. Например, инструкция CPDR. Когда Z80A выполняет ее, то:

1) Содержимое BC уменьшается на 1;

2) Уменьшается содержимое HL (о регистровой паре HL см. ниже);

3) Сравнивается содержимое аккумулятора A с содержимым ячейки памяти, адрес которой находится в паре регистров HL.

Процессор повторяет эти действия до тех пор, пока не будет совпадения между содержимым A и ячейки памяти, либо пока не обнулится BC, т.о. эта инструкция может быть использована для поиска ячейки, содержащей определенное число.

Адресные регистры "DE" и "HL"

Регистры D и E не имеют особых функций и в основном используются в качестве

временной быстро доступной памяти, они также используются в паре для хранения адреса ячейки, которая интересует нас в текущий момент.

Основная функция регистров H и L - хранить в паре адрес ячейки памяти. Мы уже видели, как определенная инструкция использует пару HL с этой целью. H хранит старший байт, L - младший. Полный адрес доступен в форме:

адрес = 256* H + L,

что дает максимум 65536 возможных адресов.

Индексные регистры "IX" и "IY"

Индексные регистры IX и IY являются 16-битовыми регистрами и могут быть использованы только так в отличие от регистров BC, DE и HL, которые могут использоваться и в паре, как 16-битовые регистры, и индивидуально, как 8-битовые. IX и IY, главным образом, используются подобно паре HL.

IX и IY, кроме того, имеют одно свойство, которое недоступно для HL - они могут быть использованы со смещением S. Имеется в виду, что инструкция, которая ссылается к (IX+S), использует не ячейку памяти, адрес которой хранится в IX, а сначала смещение S прибавляется к значению в IX, чтобы получить новый адрес, и с ним и работает инструкция.

Указатель стека "SP"

Стек - это область в верхней части ОЗУ, которая используется для временного хранения содержимого пар регистров. Стек растет вниз, когда идет заполнение и поднимается вверх при его опустошении. Начало стека фиксировано в ZX SPECTRUM - оно находится непосредственно ниже ячейки, на которую указывает системная переменная RAMTOP. Вершина стека находится ниже нижней границы стека, т.к. стек увеличивается вниз, а уменьшается вверх. Адрес текущей ячейки верхней границы стека хранится в SP регистре. Передача в стек или из стека происходит с помощью инструкций PUSH и POP. Например:

```
PUSH HL
```

В этом случае процессор:

1) Уменьшает SP;

2) Копирует содержимое регистра H в ячейку, указанную указателем стека SP;

3) Уменьшает SP;

4) Копирует содержимое регистра L в ячейку, указанную указателем стека SP.

Инструкция POP выполняет противоположное действие. По этому методу последняя поступившая пара чисел, помещенная в стек, является всегда первой парой, которая снимается со стека. Это обеспечивает простой и удобный способ временного хранения содержимого регистров во время вызова подпрограмм.

Программный счетчик "PC"

Программный счетчик PC очень важный 16-битовый регистр, т.к. он хранит адрес следующей инструкции, которая должна быть выполнена. Нормальный ход работы когда инструкция выполняется следующим образом:

1) Скопировать содержимое ячейки, указанной программным счетчиком PC в специальный регистр процессора.

2) Если инструкция содержится в нескольких байтах, увеличить PC и скопировать содержимое следующей ячейки во второй специальный регистр.

3) Увеличить PC таким образом, чтобы он указывал на следующую инструкцию, которая должна быть выполнена.

4) Выполнить инструкцию, которая только что была прочитана. Команда JUMP так же, как и DJNZ n или JR z,n , изменяет нормальный ход выполнения программы с помощью изменения PC во время выполнения шага 4. Заметим, что это изменение имеет место после увеличения PC. Таким образом, значение смещения n всегда будет отсчитываться относительно позиции инструкции, следующей за инструкцией, содержащей смещение.

Альтернативные регистры AF',BC',DE',HL'

Z80A имеет копию каждого из A, B, C, D, E, H и L регистров. Отличаются копии использованием знака " ' " возле обозначения регистра. Например, A' - это копия регистра

А. Инструкции не работают с этими регистрами непосредственно, но инструкции обмена позволяют вывести из использования 2 или более регистров и ввести в использование вместо них их копии.

Команды обмена выполняются очень быстро, т.к. содержимое регистра физически не копируется из одного регистра в другой, а вместо этого установка внутренних коммуникаций процессора изменяется так, что дублирующий регистр становится основным, а оригинальный регистр запасным.

О системе команд процессора Z80

В системе команд Z80 около 700 инструкций. Поскольку возможны только 256 различных комбинаций из 8 битов (2^8=256), то лишь менее половины команд может быть выражено одним байтом. Оставшиеся инструкции хранятся в двух или трех байтах. Некоторые команды следуют с однобайтовым смещением S, или однобайтовым числом n, или двухбайтовым числом (адресом) nn, к которым инструкция ссылается. Всего же одна инструкция может занять максимум 4 байта.

* * *

ПРИМЕЧАНИЕ "ИНФОРКОМа"

ZX SPECTRUM располагает еще встроенным программируемым калькулятором, имеющим свою систему команд. Доступ к ним возможен только из машинного кода. Команды калькулятора могут быть более длинными, чем команды процессора и занимать более 4-х байт. (См. т. 1 нашего трехтомника).

* * *

Для удобства работы на ассемблере все команды записываются с помощью мнемоник (OP CODE). Мнемоника - это сокращенное описание каждой команды.

Правила задания команд в ассемблере:

1. Одиночные регистры описываются буквой, например - B. Пара регистров именуется в алфавитном порядке, например, BC.

2. Смещение S положительно, если оно находится в пределах от 0 до 127, и отрицательно, если оно находится в пределах от 128 до 255. Большие или меньшие числа недопустимы.

Отрицательное значение подсчитывается вычитанием S из 256. Например, команда относительного безусловного перехода:

```
JR S
```

вызывает переход вперед на 8 байтов, если S=8 и переход назад на 8 байтов, если S=248, т.к.

256-8=248.

Запомните при подсчете смещения, что переход осуществляется с адреса первого байта следующей команды.

3. Однобайтовое число n лежит в пределах от 0 до 255 включительно.

4. Двухбайтовое число или адрес представляется как nn и лежит в пределах от 0 до 65535 включительно.

5. nn, заключенное в скобки, т.е. (nn), подразумевает "содержимое ячейки по адресу nn", тогда как nn без скобок подразумевает "число nn".

Т.о.,

```
LD HL,(23627)
```

подразумевает загрузку пары регистров HL содержимым ячеек 23627 и 23628, тогда как:

```
LD HL,23627
```

подразумевает загрузку HL числом 23627.

Таким же образом, (HL) подразумевает "содержимое ячейки с адресом, хранящимся в HL", тогда как HL без скобок означает "число, хранящееся в HL".

Такой вид адресации называется косвенной адресацией.

6. Место назначения результата операции всегда задается первым, например:

```
ADD A,B
```

означает "прибавить содержимое регистра В к содержимому регистра А и результат оставить в регистре А".

РАЗДЕЛ В

4. ВВЕДЕНИЕ

В Разделе В представлены программы в машинном коде. Для простоты и удобства использования они даны в стандартном формате. Во введении описывается этот формат и дается программа на BASICе, которая может быть использована для загрузки программ в память.

Длина: Это длина программы в байтах.

Количество: Выполнение некоторых программ может потребовать переменных: изменения значения одной или более переменных, передаваемых в программу через буфер принтера.

Контрольная сумма: Каждая программа дана в виде последовательности целых положительных чисел, помещаемых в последовательные ячейки памяти. Контрольная сумма (т. е. сумма всех чисел вводимой подпрограммы) дается для того, чтобы Вы были уверены в правильности загрузки

Функция: Дается краткое описание задачи, выполняемой с помощью программы.

Переменные: Определяются имя, длина и адрес в буфере принтера каждой переменной. Переменная длиной в один байт (целое положительное число в пределах от 0 до 255) передается в программу из BASICа или с клавиатуры через POKE.

POKE ячейка, значение

Двухбайтовая переменная передается с помощью 2-х команд:

```
POKE ячейка, значение - 256*INT(значение/256):
POKE ячейка+1,INT(значение/256)
```

Используемые ячейки являются ячейками памяти буфера принтера

Вызов подпрограммы: Программы вызываются с использованием функции USR, которая должна быть включена в команду. Если программа в машинном коде не передает какое-то значение обратно в BASIC по завершению, то используется команда:

```
RANDOMIZE USR адрес
```

Если Вам надо, чтобы результат был возвращен в регистровой паре ВС, то вызов делается так:

```
LET A = USR адрес
```

или

```
PRINT USR адрес
```

в зависимости от того, должны ли возвращаемые данные сохраняться в переменной BASICа или выводиться на экран.

Контроль ошибок: Объясняются проверки, выполняемые программой для нелогичных или противоречивых значений переменных, параметров и т. п.

Комментарии:	Объясняются возможные варианты в программах.
Листинг в машинном коде:	Программы представлены на языке ассемблера. Для загрузки в память используется третья колонка - "числа для ввода". Все числа здесь даны в десятеричной системе.
Как она работает:	Объяснение принципа работы программы.

Загрузчик машинного кода

Почти все программы из этой книги перемещаемые, т.е. они будут работать корректно независимо от того, в каком месте RAM мы их поместим. Если программа не перемещаемая, то в комментариях объясняется, как она должна быть изменена, если ее нужно сохранить в другой области памяти. В Разделе А (часть 2) мы видели, что SPECTRUM использует различные части RAM для различных функций и что область между ячейками, указанными системными переменными RAMTOP и UDG предназначена для хранения подпрограмм в машинном коде.

Программа BP может быть использована для загрузки, изменения и перемещения программы в машинном коде. С ее помощью пользователь может переустановить указатель RAMTOP, что даст больше свободного пространства для машинных кодов; ввести программу с клавиатуры; перейти вперед или назад для корректировки ошибки; вставить или удалить часть программы.

Когда программа BP запускается, она печатает младший адрес, с которого программа в машинных кодах может быть введена и сохранена, т.е. на единицу больший, чем RAMTOP.

В машине с 48K памяти младший адрес - 65368, если пользователь не обновлял системную переменную RAMTOP. В конце ОЗУ обычно резервируются 168 байтов для UDG, но программа позволяет пользователю использовать и эту область, если он пожелает. Он может также выбрать новый возможный младший адрес, который программа затем помещает в системную переменную RAMTOP, используя команду CLEAR. Данные не могут быть введены по адресу, меньшему, чем 27000, т.к. иначе нарушатся границы области, требующейся для самой программы BP. Программа BP запрашивает адрес, с которого должна стартовать программа в машинных кодах. Т.о. пользователь может резервировать область для нескольких процедур и затем загружать их каждую отдельно.

Рисунок BF1 показывает формат дисплея после того, как с ячейки 32000 была загружена программа "screen invert". Первая колонка - адрес, вторая - содержимое ячейки памяти с этим адресом, третья - контрольная сумма. Программа "screen invert" - имеет 18 байтов в длину и ее контрольная сумма = 1613. Следовательно, она занимает ячейки от 32000 до 32017, и контрольная сумма дана для ячейки 32017, т. е. сумма содержимого ячеек (32000...32017) равна 1613.

На основном экране внимание пользователя привлекается к одной ячейке - содержимое этой ячейки мерцает. Эта ячейка является текущей и первоначально это выбранный стартовый адрес программы. Пользователь вводит целое число между 0 и 255 включительно, которое программа MC LOADER помещает в текущую ячейку. Затем следующий адрес становится текущей ячейкой. Пользователь может не вводить число, а вместо этого выбрать для корректировки вариант, описанный в таблице BT1.

Код	Вариант
B	Перейти на один адрес назад.
B n(число)	Перейти на n адресов назад.
F	Перейти на один адрес вперед.
F n(число)	Перейти на n адресов вперед.
I n(число)	Вставить n байтов, каждый из которых содержит 0.
D n(число)	Удалить n байтов в текущей области.
T	Закончить программу

Таблица BT1. Возможные варианты редактирования машинного кода

Программа BP. загрузчик машинного кода. (MC LOADER)

```
100 GO SUB 8100
200 REM ***** Вычисление доступной памяти
210 LET min= 1+PEEK 23730+256*PEEK 23731
220 LET P = PEEK 23732+ 256*PEEK 23733
230 LET t = P - min + 1
400 REM ***** Определение стартового адреса
410 PRINT "Lowest possible start - ";min,,,"Maximum space availbie = ";t
420 INPUT "Do you wish to change the lowest start address (Y or N) ?";z$
430 IF Z$="Y" OR Z$="y" THEN GO TO 7000
440 INPUT "Enter address at which to start loading machine code";a
450 IF a<min OR a>p THEN BEEP .2,24: GO TO 440
500 GO SUB 8100
510 LET t=t-a+min
520 PRINT "You can use up to ";t;" bytes",,,
530 LET U=PEEK 23675+256*PEEK 23676
540 IF a<u AND u<p THEN PRINT "If you use more than ";u-a;" bytes, you will overwrite the
     user defined graphics area. "
550 IF a>=U THEN PRINT "You will overwrite the user defined graphics area."
560 INPUT "Is that OK (Y or N) ? ";z$
570 IF Z$="N" OR z$="n" THEN GO TO 7000
580 IF Z$<>"Y" AND z$<>"y" THEN BEEP .2,24: GO TO 560
700 REM *** GO AHEAD AND LOAD
710 LET i=a
750 GO SUB 8200
760 INPUT "Enter number, b, f, i, d or t "; z$
770 IF z$="" THEN BEEP .2,24: GO TO 760
780 LET a$=CHR$(CODE Z$(i) - 32*(Z$(i)>"@"))
790 GO TO 800+200*(a$="B")+300*(a$="F")+400*(a$="I")+500*(a$="D")+600*(a$="T")
800 LET X=VAL Z$
810 IF i>p THEN BEEP .2,24: GO TO 750
820 IF X<0 OR X>256 OR X<>INT X THEN BEEP .2,24: GO TO 760
830 POKE i,X
840 LET i=i+1
850 GO TO 740
1000 REM *** Перемещение вперед
1010 LET i=i-1
1020 IF LEN Z$>1 THEN LET i=i+1-VAL Z$(2 TO)
1030 IF i<a THEN LET i=a
1040 GO TO 740
1100 REM *** Перемещение назад
1110 LET i=i+1
1120 IF LEN Z$>1 THEN LET i=i-1+VAL Z$(2 TO)
1130 IF i>p THEN LET i=p
1140 GO TO 740
1200 REM *** Вставка
1210 IF LEN Z$=1 THEN LET n=1: GO TO 1225
1220 LET n = VAL Z$(2 TO): IF n<1 OR n>p-1 OR n<>INT n THEN BEEP .2,24: GO TO 740
1225 CLS: GO SUB 8100: PRINT TAB 6; "Inserting in Progress"
1230 FOR J=p TO i+n STEP -1
1240 POKE J,PEEK (j-n)
1250 NEXT J
1260 FOR J=1 TO i+n-1
1270 POKE J,0
1280 NEXT J
1290 GO TO 740
1300 REM *** Удаление
1310 IF LEN z$=1 THEN LET n=1: GO TO 1330
1320 LET n=VAL z$(2 TO): IF n<l OR n>P-1 OR n<>INT n THEN BEEP .2,24: GO TO 740
1330 IF n<0 OR n>p-1 THEN BEEP .2,24: GO TO 1320
1340 CLS: GO SUB 8100 :PRINT TAB 6; "DELETING IN PROGRESS"
1350 FOR J=1 TO p-n
1360 POKE J,PEEK (j+n)
1370 NEXT J
1380 GO TO 740
1400 STOP
```

```
1401 PRINT AT 21,7;"PROGRAM TERMINATED"
1410 STOP
7000 REM *** Изменение RAMTOP
7010 INPUT "ENTER NEW START ADDRESS ";a
7020 IF a<27000 OR a>p THEN BEEP .2,24: GO TO 710
7030 CLEAR a-1
7040 RUN
7999 STOP
8100 CLS
8110 PRINT TAB 6; "MACHINE CODE LOADER",,,
8120 RETURN
8200 REM *** Вывод содержимого памяти
8210 GO SUB 8100
8220 PRINT "ADDRESS DECIMAL CHECK SUM"
8230 LET c=0
8240 LET s=i-8 : IF s<a THEN LET s=a : GO TO 8280
8250 FOR j=a TO s-i 8260 LET c=c+PEEK j
8270 NEXT j
8280 LET f=s+17 : IF f>p THEN LET f=p
8290 FOR j=s TO f
8300 LET c=c+PEEK j
8310 PRINT AT j-s+3,i;j: TAB 12:PEEK j; TAB 22;c
8320 NEXT j
8400 LET pos=i-s+3
8410 PRINT AT pos,12; FLASH 1; PEEK i
8420 RETURN
```

MACHINE CODE LOADER

ADDRESS	DECIMAL	CHECK SUM
32000	33	33
32001	0	33
32002	64	97
32003	1	98
32004	0	98
32005	24	122
32006	22	144
32007	255	399
32008	122	521
32009	150	671
32010	119	790
32011	35	825
32012	11	836
32013	120	956
32014	177	1133
32015	32	1165
32016	247	1412
32017	201	1613

Рис. ВР1.

Представлен экран во время работы загрузчика машинного кода - программа "Screen Invert" загружена с адреса 32000.

5. Подпрограммы сдвига.

5.1 Сдвиг атрибутов влево.

Длина: 23
Количество переменных: 1
Контрольная сумма: 1574

Назначение: эта программа сдвигает атрибуты всех символов экрана влево на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в крайнюю правую колонку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Нет

Комментарий: Эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 24* должно быть изменено на 22.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,22528         33    0     88
            LD A,(23296)        58    0     91
            LD C,24             14    24*
NEXT_L      LD B,31             6     31
NEXT_C      INC HL              35
            LD E,(HL)           94
            DEC HL              43
            LD (HL),E           115
            INC HL              35
            DJNZ NEXT_C         16    249
            LD (HL),A           119
            INC HL              35
            DEC C               13
            JR NZ,NEXT_L        32    242
```

Как она работает:

В пару регистров HL загружается адрес области атрибутов. Аккумулятор загружается значением атрибута, вводимым в правую колонку. В регистр C загружается количество строк для сдвига - он теперь может быть использован, как счетчик строк. В регистр B заносится число на 1 меньшее, чем число символов в строке, чтобы он использовался, как счетчик. HL увеличивается, чтобы указать на следующий атрибут, который загружается в E-регистр. HL уменьшается и по адресу HL помещается значение из E-регистра. HL увеличивается вновь, чтобы указать на следующий атрибут. Регистр B уменьшается и, если он не равен 0, то происходит переход к NEXT_C'. HL теперь указывает на правую колонку и по адресу HL помещается значение из аккумулятора. HL увеличивается, чтобы указать на следующую строку - NEXT_L. Счетчик строк (регистр C) уменьшается. Если значение результата не равно 0, подпрограмма возвращается назад к NEXT_L.

По окончании работы программа возвращается в BASIC.

5.2 Сдвиг атрибутов вправо.

Длина: 23

Количество переменных: 1

Контрольная сумма: 1847

Назначение: Эта программа сдвигает атрибуты всех символов экрана вправо на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в крайнюю левую колонку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: Эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 24* должно быть изменено на 22.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

           LD HL,23295        33    255   88
           LD A,(23296)       58    0     91
           LD C,24            14    24*
NEXT_L     LD B,31            6     31
NEXT_C     DEC HL             43
           LD E,(HL)          94
           INC HL             35
           LD (HL),E          115
           DEC HL             43
           DJNZ,NEXT_C        16    249
           LD (HL),A          119
           DEC HL             43
           DEC C              13
           JR NZ,NEXT_L       32    242
           RET                201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта области атрибутов. Аккумулятор загружается значением атрибута для ввода в левую колонку. В регистр C загружается количество строк для сдвига - он используется, как счетчик строк. В регистр B заносится число на 1 меньшее, чем число символов в строке для использования его в качестве счетчика.

HL увеличивается, чтобы указать на следующий атрибут. Значение этого атрибута загружается в E-регистр. HL увеличивается, и по адресу HL помещается значение из E-регистра. HL уменьшается снова для указания адреса следующего атрибута. Счетчик в регистре B уменьшается и, если он не равен 0, то - возврат назад к NEXT_C. HL теперь указывает на крайнюю левую колонку, и в ячейку с адресом HL, помещается значение из аккумулятора. HL уменьшается для указания правого конца следующей строки. Счетчик строк уменьшается, и если он не равен 0, - возврат назад к NEXT_L.

Программа возвращается в BASIC.

5.3 Сдвиг атрибутов вверх.

Длина: 21

Количество переменных: 1

Контрольная сумма: 1591

Назначение: Эта программа сдвигает атрибуты всех символов экрана вверх на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в нижнюю строку.

Вызов программы:

```
RAMDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 224* должно быть изменено на 160.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22560        33    32    88
            LD DE,22526        17    0     88
            LD BC,736          1     224*  2
            LDIR               237   176
            LD A,(23296)       58 0  91
            LD B,32 6          32
NEXT_C      LD (DE),A          18
            INC DE             19
            DJNZ NEXT_C        16    252
            RET 201
```

Как она работает

В пару регистров HL загружается адрес второй строки атрибутов, в DE загружается адрес первой строки, и BC загружается числом байтов для перемещения.

Количество байтов, определенное в регистровой паре BC, копируются в ячейки памяти с адресом, находящимся в DE из ячеек, начинающихся с адреса в HL, используя инструкцию LDIR. Эти результаты в DE указывают на нижнюю строку атрибутов. Аккумулятор загружается кодом атрибута для ввода в нижнюю строку. B-регистр загружается числом символов на одной строке - он используется, как счетчик.

По адресу DE помещается значение из аккумулятора, а затем DE увеличивается для указания на следующий байт. Счетчик уменьшается и, если он не равен 0, подпрограмма возвращается к NEXT_C.

Затем программа возвращается в BASIC.

5.4 Сдвиг атрибутов вниз.

Длина: 21

Количество переменных: 1

Контрольная сумма: 2057

Назначение: эта программа сдвигает атрибуты всех символов экрана вниз на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарии: это атрибут, вводимый в верхнюю строку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: эта программа полезна для выделения области текста или графики, для прокручивания только 22 верхних строк должны быть изменены: 223* на 159 255* на 191 224* на 160

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,23263        33    223   90
            LD DE,23295        17    255   90
            LD BC,736          1     224   2
            LDDR               237   184
            LD A,(23296)       58    0     91
            LD B,32            6     32
NEXT_C      LD (DE),A          18
            DEC DE             27
```

```
          DJNZ NEXT_C         16    252
          RET                 201
```

Как она работает:

В HL загружается адрес последнего атрибута 23-й строки. В DE загружается адрес последнего атрибута 24-й строки. В ВС загружается число байтов для перемещения, затем команда LDDR перемещает байты (их количество указано в регистровой паре ВС) из адреса в HL по адресу в DE. Эти результаты в DE хранят адрес последнего атрибута первой строки.

В аккумулятор загружается значение атрибута для ввода в верхнюю строку. В В регистр загружается число байтов в верхней строке - он используется, как счетчик. В ячейку с адресом DE помещается значение из аккумулятора и DE уменьшается для указания на следующий байт. Счетчик уменьшается и, если он не равен 0, подпрограмма возвращается к NEXT_C. Программа затем возвращается в BASIC.

5.5 Сдвиг влево на один символ.

Длина:21

Количество переменных:0

Контрольная сумма:1745

Назначение: эта программа прокручивает графику экрана на один символ влево. Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий:

Эта программа полезна при организации экрана в качестве "окна", показывающего в данный момент меньшую часть большой дисплейной области. Это "окно" перемещается, используя подпрограммы сдвига.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,16384         33    0     54
            LD D,L              85
            LD A,192            62    192
NEXT_L      LD B,31             6      31
NEXT_B      INC HL              35
            LD E,(HL)           94
            DEC HL              43
            LD (HL),E           115
            INC HL              35
            DJNZ NEXT_B         16    249
            LD (HL),D           114
            INC HL              35
            DEC A               61
            JR NZ,NEXT_L        32    242
            RET                 201
```

Как она работает:

В пару регистров HL загружается адрес дисплейного файла, а D-регистр устанавливается в 0. В аккумулятор загружается число строк на экране. В В-регистр загружается значение на 1 меньшее, чем число символов в строке - оно является числом байтов для копирования.

HL увеличивается для указания на следующий адрес и содержимое из этой ячейки загружается в Е-регистр. HL уменьшается и в ячейку с адресом HL загружается значение из Е-регистра. HL увеличивается для адресации следующего байта и счетчик в В-регистре уменьшается. Если он не равен 0, подпрограмма возвращается к NEXT_B.

Если регистр В равен 0, это означает, что последний байт строки скопирован и HL указывает на крайний правый байт. По этому адресу в регистровой паре HL помещается 0 и

HL увеличивается, указывая на следующую строку. Счетчик строк - аккумулятор - уменьшается, и, если он не равен нулю, происходит переход к NEXT_L.

Программа возвращается в BASIC.

5.6 Сдвиг вправо на один символ.

Длина:22

Количество переменных:0

Контрольная сумма:1976

Назначение: эта программа прокручивает содержимое дисплейного файла на один символ вправо.

Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий:

Эта программа полезна при организации экрана в качестве "окна", показывающего в данный момент меньшую часть большой дисплейной области. Это "окно" перемещается, используя подпрограммы сдвига.

Листинг машинных кодов

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22527        33    255   87
            LD D,0             22    0
            LD A,192           62    192
NEXT_L      LD B,31 6          31
NEXT_B      DEC HL             43
            LD R,(HL)          94
            INC HL             35
            LD (HL),E          115
            DEC HL             43
            DJNZ NEXT_B        16    249
            LD (HL),D          114
            DEC HL             35
            DEC A              61
            JR NZ,NEXT_L       32    242
            RET                201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта дисплейного файла, а D-регистр устанавливается в 0. В аккумулятор загружается число строк на экране. В B-регистр загружается значение на 1 меньшее, чем число символов в строке, - он используется, как счетчик.

HL уменьшается, указывая на следующий байт, и его значение загружается в E-регистр. HL затем увеличивается, и в ячейку с адресом HL загружается значение E-регистра. HL уменьшается, указывая на следующий байт, и счетчик (B-регистр) уменьшается. Если он не равен 0, подпрограмма возвращается к NEXT_B.

Если регистр B равен 0, это означает, что HL указывает на крайний левый байт строки. Затем по адресу в регистровой паре HL помещается 0, и HL уменьшается, указывая на следующую строку. Счетчик строк (аккумулятор) уменьшается и, если он не равен 0, происходит переход к NEXT_L.

Программа возвращается в BASIC

5.7 Сдвиг вверх на один символ

Длина: 68

Количество переменных:0

Контрольная сумма: 6326

Назначение: программа сдвигает содержимое дисплейного файла вверх на восемь

пикселей.

Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: нет

Листинг машинных кодов

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ  ВВОДА

            LD HL,16364        33     0      64
            LD DE,16416        17     32     64
SAVE        PUSH HL            229
            PUSH DE            213
            LD C,23            14     23
NEXT_L      LD B,32            6      32
COPY_B      LD A,(DE)          26
            LD (HL),A          119
            LD A,C             121
            AND 7              230    7
            CP 1               254    1
            JR NZ,NEXT_B       32     2
            SUB A              151
            LD (DE),A          18
NEXT_B      INC HL             35
            INC DE             19
            DJNZ COPY_B        16     241
            DEC C              13
            JR Z,REST          40     19
            LD A,C             121
            AND 7              230    7
            CP 0               254    0
            JR Z,N_BLOCK       40     22
            CP 7               254    7
            JR NZ,NEXT_L       32     225
            PUSH DE            213
            LD DE,1792         17     0      7
            ADD HL,DE          25
            POP DE             209
            JR NEXT_L          24     217
REST        POP DE             209
            POP HL             225
            INC D              20
            INC H              36
            LD A,H             124
            CP 72              254    72
            JR NZ,SAVE         32     204
            RET                201
N-BLOCK     PUSH HL            229
            LD HL,1792         33     0      7
            ADD HL,DE          25
            EX DE,HL           235
            POP HL             225
            JR NEXT_L          24     198
```

Как она работает:

В пару регистров HL загружается адрес начала дисплейного файла, а в DE загружается адрес байта через восемь линий вниз. HL и DE сохраняются в стеке, в C-регистр загружается число на 1 меньшее, чем число строк на экране. В B-регистр загружается количество байтов на одной линии дисплея - он используется, как счетчик.

В аккумулятор загружается байт, адресованный DE, и это значение загружается в ячейку по адресу HL. В аккумулятор загружается содержимое C-регистра и, если оно равно

1, 9 или 17, то в ячейку по адресу DE помещается 0. HL и DE увеличиваются, указывая на следующий байт, счетчик в B-регистре уменьшается и, если он не равен 0, происходит переход к 'COPY_B'.

Счетчик строк в регистре C затем уменьшается. Если он равен 0, происходит переход к 'RESTORE'. Если C содержит 8 или 16, то происходит переход к 'N_BLOCK'. Если C не содержит 7 или 15, подпрограмма переходит к 'NEXT_L'. Затем 1792 прибавляется к HL - теперь HL указывает на следующий блок экрана. Подпрограмма переходит к 'NEXT_L'.

В процедуре 'REST' DE и HL берутся из стека и 256 прибавляется к каждому из них. Т.о., DE и HL указывают на строку, позиция которой ниже, чем та, что была в предыдущем цикле. Если HL содержит значение 18432, подпрограмма возвращается в BASIC, иначе происходит переход к процедуре 'SAVE'. В процедуре 'N_BLOCK' 1792 прибавляется к DE - таким образом DE указывает на следующий блок экрана. Подпрограмма затем возвращается к 'NEXT_L'.

По окончании работы происходит возврат в БЕЙСИК.

Продолжение следует

Алексеев А. Г.

МАСТЕРФАЙЛ 09 полная русификация.

ВСТУПЛЕНИЕ.

Прежде всего я хотел бы сказать что эта статья ориентирована на тех пользователей "Спектрума" которые уже немного овладели программированием на Бейсике, зашитом в ПЗУ компьютера но, еще не владеют знаниями, необходимыми для программирования в машинных кодах Z80. Но, пользуясь только этими знаниями, многие усовершенствования можно уже делать с готовыми программами. А стимул для дальнейшего освоения машинных кодов несомненно появится в процессе такой работы. Таким образом, психологический барьер будет преодолён и Вы выйдете на новый уровень.

В этой статье изложены основные методы и приемы неполной и полной русификации программ для "Спектрума". В качестве примера будут рассмотрены способы русификации программы "MF 09". Она достаточно широко распространена среди пользователей "Спектрума" и английский вариант сильно сдерживает ее применение.

Существует несколько способов русификации "Спектрума". Сначала немного о самом простом из них.

1. Использование символов UDG-графики.

Так как многие латинские буквы в режиме CAPS LOCK по написанию совпадают с русскими (А, В, Е, К, М, Н, О, Р, С, Т, Х), а вместо русской буквы "З" можно использовать цифру "3", то остается всего 20 букв, требующихся для русификации (Б, Г, Д, Ж, И, Й, А, П, У, Ф, Ц, Ч, Ш, Щ, Ъ, Ы, Ь, Э, Ю, Я), то есть мы укладываемся в 21 символ, которые отведены для графики пользователя в "Спектруме".

Предлагаемая программа наглядно показывает принцип формирования символа UDG-графики. Например, для формирования буквы "Я", закрепленной за символом UDG "А", набираем программу:

```
10 LET n=USR "a"
20 FOR x=n TO n+7
30 READ y
40 POKE x,y
50 NEXT  x
100 DATA
       BIN 00000000,
       BIN 00111110,
       BIN 01000010,
       BIN 01000010,
       BIN 00111110,
       BIN 00100010,
       BIN 01000010,
       BIN 00000000
```

В строке 100 после запятых надо набирать по 19 пробелов, чтобы наглядно просматривалась будущая буква. Те пиксели, которые должны быть включены, отмечаем как "1", а те, которые выключены - как "0".

Для буквы "Б" надо в строке 10 вместо USR "a" подставить USR "b" и изменить нули и единицы в строке 100.

Более подробно этот метод был изложен в разработке ИНФОРКОМА "Большие возможности Вашего Спектрума". Говорилось о нем и в ZX-РЕВЮ N 4-5 (стр. 80). Поэтому,

чтобы не повторяться, представим теперь, что русские UDG-символы Вами уже сформированы и Вы записываете 21 символ, начиная с "a", на магнитофон:

```
SAVE "rusudg" CODE USR "a",21*8
```

При наборе своих собственных программ можно поступать, например, следующим образом. После рестарта компьютера набрать:

```
1 GO TO 100
2 LOAD "rusudg"  CODE
3 GO TO 1
5 SAVE  "zagotowka" LINE 2:
SAVE "rusudg"  CODE USR "a",168
6 GO TO 5
```

Получился своеобразный "дебют" программы. Теперь сделайте RUN 2 и загрузите записанную ранее область UDG. После сообщения "0 OK" набирайте свою программу, начиная с той строки, которую указали в строке 1 после GO TO, то есть с 100. Для записи готовой программы на магнитофон используйте RUN 5, предварительно подставив вместо "zagotowka" имя программы. При этом запишется программа, а следом за ней блок кодов UDG. Строка 6 "зацикливает" процесс записи, если Вам надо записать несколько дублей программы. При загрузке программа автостартует со строки 2 и загрузит блок UDG. Команда RUN обеспечивает "холодный" старт программы.

Блок кодов UDG-графики, также, как и любой другой блок кодов, можно разместить внутри Бейсик-программы, используя для этого нулевую строку. Это имеет определенные преимущества: программа состоит не из двух, а из одного куска, сокращается время загрузки. Делается это очень просто.

После рестарта компьютера наберите: 1 REM, а после REM столько пробелов (или любых других символов), сколько байтов памяти Вам надо зарезервировать для Ваших целей. Для размещения, например, блока символов UDG надо набрать 168 пробелов. После того, как строка введена в память компьютера, заменяем ее номер на 0, подав прямую команду: POKE 23756,0 . На экране теперь видим: 0 REM. Первая строка стала нулевой. Теперь ее невозможно случайно испортить, вызвав на редактирование командой EDIT. Свободная область в этой строке начинается с первого символа (пробела), стоящего за REM, адрес ее начала - 23760, длина равна числу набранных символов (пробелов). Теперь, если Вы набрали 168 пробелов, можете загрузить в нулевую строку блок кодов UDG-графики:

```
LOAD "rusudg" CODE 23760
```

Далее надо сделать так, чтобы включался блок UDG-кодов, загруженный в новое место. Для этого используем системную переменную UDG, занимающую два байта в ячейках 23675 и 23676. Эта переменная указывает адрес первой ячейки области UDG. Выполните:

```
PRINT PEEK 23675+256*PEEK 23676
```

Вы получите результат: 65368. Кстати, этот же результат Вы получите, выполнив: PRINT USR "a".

Новое значение системной переменной UDG будет равно 23760. Вычислим младший, а затем старший байты этого числа, выполнив:

```
PRINT 23760-256*INT(23760/256)
PRINT INT(23760/256)
```

Получим 208 и 92. Теперь догрузим к нулевой строке наш "дебют" программы, выполнив:

```
MERGE "zagotowka"
```

и изменим строки 2 и 5:

```
1 GO TO 100
2 POKE 23675,208: POKE 23676,92
3 GO TO 1
5 SAVE "zasotowka" LINE 2
```

```
6 GO TO 5
```

Теперь сделайте RUN 2. После сообщения "0 OK" можете набирать свою программу, используя графический регистр. Надо только помнить о том, что при выполнении команды LIST при выведении нулевой строки, скорее всего, будет сообщение об ошибке. Это связано с тем, что интерпретатор Бейсика не понимает ту информацию, которая стоит в нулевой строке. Для получения листинга придется делать LIST 1. Неприятности могут быть также и в том, что не все начальные строки могут быть видны в автоматическом листинге, хотя они нормально вызываются при редактировании командой EDIT и работа готовой программы от этого никак не страдает.

Для несложных своих программ этот способ русификации вполне может применяться, так как позволит писать на русском языке даже комментарии в тексте программы.

Что касается программы MF 09, то, к сожалению, к ней не подходит этот самый простой способ русификации, так как символы, набранные с использованием графического регистра (курсор [G]) не отображаются в режиме "дисплей". Вместо них на экране печатается знак "?". Такие же трудности возникнут и при выведении текста на печать при помощи принтеров, отличных от "ZX". Поэтому рассмотрим другой, более совершенный способ неполной русификации.

2. Использование дополнительного символьного набора.

Некоторые проблемы, связанные с этим вопросом, были изложены в ZX-РЕВЮ № 4-5 стр. 80-81. Наиболее крупная из всех - это, по моему, программная совместимость. Какой латинской букве будет соответствовать какая русская? Это не имеет большого значения для конкретной игровой программы, где не надо вводить текст в процессе игры, однако для таких программ, как MF 09, это имеет немаловажное значение, ведь со временем все больше и больше появится возможностей пользоваться базами данных, составленными другими авторами. Это могут быть и каталоги программ для "Спектрума" (с комментариями на русском языке), и расписание движения поездов, самолетов и т.д. Даже если Вы вместе с базой данных перепишете и саму программу MF 09 автора, трудности возникнут при внесении дополнений и изменений в базу данных, так как каждый раз надо будет "осваивать" незнакомое распределение русских букв между кнопками клавиатуры.

К стандарту кодов ASCII автор этих строк подошел своим, независимым от ИНФОРКОМА путем, но результат все равно тот же. Значит наверняка есть и другие авторы, работающие в этом стандарте. Кстати и символьный набор "Спектрума" (коды с 32 по 127) являются кодами ASCII. Поэтому настоятельно рекомендую начинающим придерживаться именно этого стандарта. Распределение русских букв на первый взгляд может показаться неудобным, но Вы быстро привыкнете и перестанете это неудобство замечать. Зато у Вас будет больше шансов найти "привычные" для себя программы.

Базисная таблица КОИ-7 кодов ASCII имеет несколько символьных наборов: латинский, русский, смешанный. Символьный набор условно можно разделить на три части:

1 часть - коды с 32 по 63 - символы и цифры;

2 часть - коды с 64 по 95 - буквы, в основном печатаемые в регистре CAPS LOCK;

3 часть - коды с 96 по 127 - буквы, печатаемые без регистра CAPS LOCK.

Вы можете увидеть эти три части в трех строках на экране, выполнив:

```
FOR a=32 TO 127: PRINT CHR$ a;: NEXT a
```

То, что Вы видите на экране, является символьным набором "Н0" базисной кодовой таблицы КОИ-7 кодов ASCII. Если взять другой символьный набор: "Н1", то у него первая часть остается той же, вторая часть - это строчные русские буквы, а третья часть - заглавные русские буквы. Таблица их соответствия с тем набором, который в ROM "Спектрума" приведена в ZX-РЕВЮ №4-5 за 1991г., стр. 81.

Когда Вы принципиально решили для себя вопрос стандартизации, все остальное - дело техники. Загружаем программу "ART STUDIO", входим в режим TEXT и далее FONT EDITOR. Теперь надо переделать латинские буквы на русские соответственно таблице (вторую и третью части символьного набора), оставив без изменения цифры и символы (первую часть символьного набора) и записать новый набор символов на ленту (например с

именем "ruschr").

Полученный символьный набор занимает 768 байтов в памяти. Для использования в своих программах, его удобно расположить непосредственно перед символами UDG, разместив с адреса:

```
USR "a"-768=64600
```

Кстати, сохранять на ленте целесообразно символьный набор вместе с блоком кодов UDG; последние могут использоваться в Ваших программах непосредственно по назначению - для получения графических элементов. Для этого надо обьединить эти блоки в один:

```
LOAD "ruschr" CODE 64600,768
LOAD "rusudg" CODE 65368,168
SAVE "rus" CODE 64600,936
```

Теперь подумаем об удобстве пользования. Если символьный набор расположен с адреса 64600, то системная переменная CHARS (ячейки 23606, 23607) будет равна:

64600-256=64344

Младший и старший байты ее:

64344-256*INT(64344/255)=88

INT(64344/256)=251

"Дебют" программы может выглядеть теперь следующим образом:

```
1 GO TO 100
2 LOAD ""
3 GO TO 1
5 SAVE "zagotowka" LINE 2 : SAVE "rus" CODE 64600,936
6 GO TO  5
8 POKE 23606,88: POKE 23607, 251: RETURN : REM rus
9 POKE 23606,0 : POKE 23607,60 :RETURN : REM lat
```

Для записи готовой программы, как и раньше, используется RUN 5 (предварительно подставив имя программы). Следом за программой будет записан символьный набор с блоком UDG (т.е. 768+168=936 байт). Теперь поговорим о строках 8 и 9. Подайте прямые команды:

```
GO SUB 8 и затем: LIST
```

Вы увидите, что листинг программы печатается русскими буквами. Для обратного переключения сделайте GO SUB 9. Теперь эти переключения можно применять в Вашей программе. Например, чтобы написать фразу: "ZX-Spectrum пишет ПО-РУССКИ", надо ввести такую строку в программе:

```
100 PRINT "ZX-Spectrum ";:GO SUB 8: PRINT "pi{et PO-RUSSKI":GO SUB 9
```

Чтобы не допускать ошибок при наборе большого обьема русского текста в своей программе, перед набором строки сделайте GO SUB 8 прямой командой и Вы будете видеть русский текст, который набиваете, в том виде, как он будет напечатан на экране.

Применение двух символьных наборов позволяет выводить на экран текст русскими и латинскими заглавными и строчными буквами, однако производить переключения вставляя в программу GO SUB 8 и GO SUB 9 не очень удобно, но главное, эти переключения нельзя выполнить, не останавливая программу, например, набирая строку данных в программе MF 09. Поэтому можно воспользоваться другим набором кодов ASCII - это набор КОИ-7 "НС". Здесь первая и вторая части - совпадают с ROM "Спектрума", а в третьей части вместо строчных латинских - заглавные русские буквы.

Таблица соответствия с ROM "Спектрума" приведена ниже (только третья часть символьного набора):

99	c	Ц	115	s	С
100	d	Д	116	t	Т
101	e	Е	117	u	У
102	f	Ф	118	v	Ж
103	g	Г	119	w	В
104	h	Х	120	x	Ь

105	i	И	121	y	И
106	j	Й	122	z	З
107	k	К	123	[	Ш
108	l	Л	184	верт. черта	Э
109	m	М	125	]	Щ
110	n	Н	126	апостроф	Ч
111	o	О	127	копирайт	Ъ

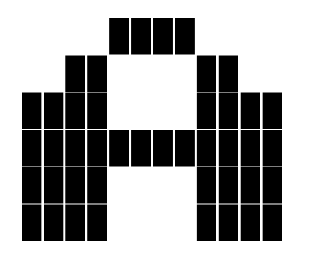
В этом случае переключение на символьный набор можно сделать только один раз сразу же после старта программы. Измените строку 3 нашего дебюта:

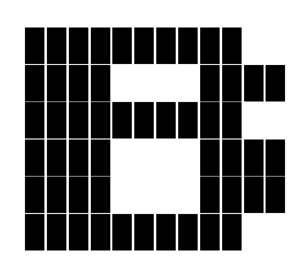
```
3 GO SUB 8: GO TO 1
```

При этом, правда, придется пользоваться хотя и русским и латинскими но только заглавными буквами.

Попутно следует сказать несколько слов о том, как выглядит готовый шрифт на экране компьютера. Вы обращали внимание, что почти ни в одной хорошей коммерческой программе не применяется символьный набор "Спектрума" а, в основном стилизованные шрифты. К особому зрительному эффекту приводит применение уплотнённых шрифтов. При этом программа буквально преображается. Почему бы не использовать такие шрифты в своих программах? Кроме красивого "внешнего вида" они имеют повышенную четкость при различном сочетании цветов INK - PAPER.

В своих программах я применяю "утолщенный" русско-латинский символьный набор ("НС" в кодах КОИ-7), который хочу предложить вашему вниманию. При этом буквы выглядят в увеличенном виде следующим образом (██ - один пиксел):





Однако большие массивы текста, состоящего только из заглавных букв, да еще "утолщенных", на экране выглядят слишком плотно и тяжело. Поэтому русские буквы в этом символьном наборе сделаны пониже, чем латинские. По высоте - как строчные буквы, а по написанию - как заглавные. При этом как бы увеличивается расстояние между строчками и текст выглядит более гармонично. Все то, о чем говорится в этих строках, может быть подвергнуто сомнению. К тому же набирать 768 чисел вручную утомительно. Но попробуйте, хотя бы из любопытства я уверен: то, что Вы получите, вам понравится. Одни явные преимущества этого символьного набора Вы увидите сразу, а другие несомненно оцените, поработав некоторое время. Мои друзья и знакомые давно с удовольствием пользуются именно этим набором, поскольку он достаточно универсален и текст замечательно выглядит на экране.

Для ввода этого символьного набора в память компьютера, надо набрать программу.

```
10 CLEAR 54599
20 LET N=64600: LET S=0
30 FOR X=N TO N+767
40 READ Y: POKE X,Y
50 LET S=S+Y:
60 NEXT X
70 IF S<>44655 THEN PRINT FLASH 1;"ERROR":STOP
80 POKE 23606,88: POKE 23607,251
90 FOR A=32 TO 127: PRINT CHR$ A;: NEXT A
100 SAVE "znak" CODE 64600,768
110 DATA
    000, 000, 000, 000, 000, 000, 000, 000,
    000, 024, 024, 024, 024, 000, 024, 000,
```

```
    000, 108, 108, 108, 000, 000, 000, 000,
    000, 108, 254, 108, 108, 254, 108, 000
120 DATA
    000, 024, 126, 088, 126, 026, 126, 024,
    000, 098, 100, 008, 016, 038, 070, 000,
    000, 048, 088, 048, 122, 204, 118, 000,
    000, 024, 048, 000, 000, 000, 000, 000
130 DATA
    000, 012, 024, 024, 024, 024, 012, 000,
    000, 048, 024, 024, 024, 024, 048, 000,
    000, 000, 108, 056, 254, 056, 108, 000,
    000, 000, 024, 024, 126, 024, 024, 000
140 DATA
    000, 000, 000, 000, 000, 024, 024, 048,
    000, 000, 000, 000, 124, 000, 000, 000,
    000, 000, 000, 000, 000, 024, 024, 000,
    000, 000, 006, 012, 024, 048, 096, 000
150 DATA
    000, 060, 102, 110, 118, 102, 060, 000.
    000, 024, 056, 024, 024, 024, 060, 000,
    000, 060, 102, 005, 060, 096, 126, 000,
    000, 060, 102, 012, 006, 102, 060, 000
160 DATA
    000, 012, 028, 044, 076, 126, 012, 000,
    000, 126, 096, 124, 006, 102, 060, 000,
    000, 060, 096, 124, 102, 102, 060, 000,
    000, 126, 102, 012, 024, 048, 048, 000
170 DATA
    000, 060, 102, 060, 102, 102, 060, 000,
    000, 060, 102, 102, 062, 006, 060, 000,
    000, 000, 024, 024, 000, 024, 024, 000,
    000, 000, 024, 024, 000, 024, 024, 048
180 DATA
    000, 000, 012, 024, 048, 024, 012, 000,
    000, 000, 000, 124, 000, 124, 000, 000,
    000, 000, 048, 024, 012, 024, 048, 000,
    000, 060, 102, 012, 024, 000, 024, 000
190 DATA
    000, 124, 206, 214, 222, 192, 124, 000,
    000, 060, 102, 102, 126, 102, 102, 000,
    000, 124, 102, 124, 102, 102, 124, 000,
    000, 060, 102, 096, 096, 102, 060, 000
200 DATA
    000, 124, 102, 102, 102, 102, 124, 000,
    000, 126, 096, 124, 096, 096, 126, 000,
    000, 126, 096, 124, 096, 096, 096, 000,
    000, 060, 102, 096, 110, 102, 060, 000
210 DATA
    000, 102, 102, 126, 102, 102, 102, 000,
    000, 060, 024, 024, 024, 024, 060, 000,
    000, 014, 006, 006, 102, 102, 060, 000,
    000, 102, 108, 120, 120, 108, 102, 000
220 DATA
    000, 096, 096, 096, 096, 096, 126, 000,
    000, 066, 102, 126, 102, 102, 102, 000,
    000, 102, 102, 116, 110, 102, 102, 000,
    000, 060, 102, 102, 102, 102, 060, 000
230 DATA
    000, 124, 102, 102, 124, 096, 096, 000,
    000, 060, 102, 102, 110, 110, 062, 000,
    000, 124, 102, 102, 124, 106, 102, 000,
    000, 060, 096, 060, 006, 102, 060, 000
240 DATA
    000, 126, 024, 024, 024, 024, 024, 000,
    000, 102, 102, 102, 102, 102, 060, 000,
```

```
     000, 102, 102, 102, 102, 060, 024, 000,
     000, 196, 198, 214, 214, 254, 066, 000
 250 DATA
     000, 102, 060, 024, 024, 060, 102, 000,
     000, 102, 060, 024, 024, 024, 024, 000,
     000, 124, 012, 024, 048, 096, 124, 000,
     000, 030, 024, 024, 024, 024, 030, 000
 260 DATA
     000, 000, 096, 048, 024, 012, 006, 000,
     000, 120, 024, 024, 024, 024, 120, 000,
     000, 024, 060, 090, 024, 024, 024, 000,
     000, 000, 000, 000, 000, 000, 000, 255
 270 DATA
     000, 000, 220, 246, 245, 246, 220, 000,
     000, 000, 060, 102, 126, 102, 102, 000,
     000, 000, 124, 096, 124, 102, 124, 000,
     000, 000, 106, 106, 108, 108, 126, 006
 280 DATA
     000, 000, 060, 108, 108, 108, 254, 196,
     000, 000, 126, 096, 124, 096, 126, 000,
     000, 000, 124, 214, 214, 124, 016, 000,
     000, 000, 062, 048, 048, 048, 048, 000
 290 DATA
     000, 000, 102, 060, 024, 060, 102, 000,
     000, 000, 102, 102, 110, 118, 120, 000,
     000, 024, 102, 102, 110, 118, 102, 000,
     000, 000, 102, 108, 120, 108, 102, 000
 300 DATA
     000, 000, 030, 054, 054, 054, 102, 000,
     000, 000, 066, 102, 126, 102, 102, 000,
     000, 000, 102, 102, 126, 102, 102, 000,
     000, 000, 060, 102, 102, 102, 060, 000
 310 DATA
     000, 000, 126, 102, 102, 102, 102, 000,
     000, 000, 062, 102, 062, 054, 102, 000,
     000, 000, 124, 102, 102, 124, 096, 000,
     000, 000, 060, 102, 096, 102, 060, 000
 320 DATA
     000, 000, 126, 024, 024, 024, 024, 000,
     000, 000, 102, 102, 062, 006, 060, 000,
     000, 000, 214, 214, 124, 214, 214, 000,
     000, 000, 124, 102, 124, 102, 124, 000
 330 DATA
     000, 000, 096, 096, 124, 102, 124, 000,
     000, 000, 198, 198, 246, 222, 246, 000,
     000, 000, 060, 102, 012, 102, 060, 000,
     000, 000, 198, 214, 214, 214, 254, 000
 340 DATA
     000, 000, 120, 012, 060, 012, 120, 000,
     000, 000, 198, 214, 214, 214, 255, 003,
     000, 000, 102, 102, 062, 006, 006, 000,
     060, 066, 153, 161, 161, 153, 066, 060
```

Числа в строках DATA напечатаны друг под другом для удобства чтения. Вы же можете набирать их так, как Вам удобно.

Теперь о "встраивании" загружаемого символьного набора в программу MF 09.

После того, как символьный набор будет записан на ленту, поступаем следующим образом. Разместим его непосредственно перед основным блоком кодов "MF09CODE" CODE 57328,8208 с адреса 57328-768-56560. Значение переменной CHARS тогда будет равно: 56560-256=56304.

Младший и старший байты CHARS:

56304-256*INT(56304/256)=240

INT (56304/256)=219

Далее делаем:

```
CLEAR 56559
LOAD "znak" CODE 56560,768
LOAD "MF09CODE" CODE 57328,8208
```

И записываем готовую программу на ленту:

```
SAVE "MF09 R/L" CODE 56560,8976
```

Теперь изменяем программу-загрузчик "MF LOADER", заменив CLEAR 57327 на CLEAR 56559.

Изменения, производимые в основной Бейсик-программе. Добавляем строки, переключающие основной и альтернативный символьные наборы

```
8 POKE VAL "23606",VAL "240": POKE VAL "23607", VAL "219": RETURN : REM RUS
9 POKE VAL "23606", NOT PI : POKE VAL "23607",VAL "60": RETURN : REM lat
```

Изменяем другие строки, подставляя в нужные места GO SUB 8 или GO SUB 9 и изменив начальный адрес и длину основного блока кодов:

```
1 GO SUB VAL "8": GO TO USR VAL "58285"
4020 GO SUB VAL "9": SAVE C$(TO VAL "10") DATA F$(): GO SUB VAL "8": GO TO USR R
4030 GO SUB VAL "9": SAVE C$(TO VAL "10") LINE VAL "4035": SAVE "MF09 R/L" CODE VAL "56560",
     VAL "8976": GO SUB VAL "8": GO TO USR R
4035 LOAD "MF09 R/L" CODE: GO TO PI/PI
```

Остальные строки оставляем без изменения.

Описанные выше способы русификации относились только к вводимым данным, однако более высокой степенью русификации является перевод на русский язык и замена текстовых сообщений в программе, которые печатаются по-английски.

3. Перевод программы на русский язык.

Пусть Вас не пугает то, что программа написана в машинных кодах. На самом деле все ненамного страшнее, чем в Бейсике. Причем для того, чтобы сделать полный перевод программы на русский язык, не обязательно даже умение пользоваться какими-либо специальными программами-мониторами типа MONS или другими. Не обязательно также знание шестнадцатеричной системы счисления. Необходимо только желание, немного терпения и аккуратности, да хоть немного знать английский язык или иметь словарь потолще. Неплохо также, если Вы поработали с программой какое то время, чтобы Вам понятен был смысл того или иного текстового сообщения.

Рассмотрим подробно процесс перевода программы на русский язык на примере программы MF 09. Приобретенный опыт Вы сможете использовать для перевода других программ.

В машинных кодах процедуры вывода текстовых сообщений на экран могут быть самыми различными, но в любом случае сама строка символов, выводимая на экран, находится внутри программы, надо только найти ее и изменить коды символов, находящихся там, занося другие значения хотя бы при помощи POKE.

Для работы можно воспользоваться любой программой-монитором, которая есть под рукой. А можно за несколько минут набрать нужную программу на Бейсике.

Ниже приводится описание такого специализированного монитора, который поможет нам находить текстовые сообщения в программе, а также несколько автоматизирует процесс замены текста на русский.

ПРОГРАММА-МОНИТОР

```
1 BORDER 7: PAPER 7: INK 0: CLS: GO SUB 6: GO TO 100
2 CLEAR 50000
3 LOAD ""CODE
4 GO TO 1
5 GO SUB 9: INPUT "FILENAME"; N$
6 SAVE N$ LINE 2: SAVE N$ CODE 56560,8976
7 STOP
8 POKE 23606,240: POKE 23607, 219: RETURN : REM rus
9 POKE 23606,0: POKE 23607,60: RETURN : REM lat
10 LET B=PEEK A: LET C$=CHR$ B
15 IF B<32 THEN LET C$="?"
```

```
20 RETURN
100 INPUT "ADDRESS: ";N
200 FOR A=N TO 65535
210 REM
220 GO SUB 10
230 INPUT (TAB 0;A; TAB 8; B;TAB 13;c$;TAB 23);C$: IF c$<>"" THEN POKE A,CODE C$
231 REM INPUT (TAB 0: A; TAB 8; B; TAB 13; C$; TAB 23); LINE C$: IF C$<>"" THEN POKE A,VAL
    C$
240 GO SUB 10
250 PRINT TAB 0; A; TAB 8;B; TAB 13; c$;
300 NEXT A
```

В строке 15 непосредственно перед знаком вопроса надо нажать "INV. VIDEO" (CAPS SHIFT+4), а сразу же после знака вопроса "TR. VIDEO" (CAPS SHIFT+3).

REM в строке 231 набран после набора всей строки, в последнюю очередь.

После того, как Вы наберете эту программу, сделайте RUN 2 и загрузите с магнитофона коды "MF09 R/L" с пристыкованным русско-латинским символьным набором. После окончания загрузки программа выполнит переключение на этот символьный набор и запросит адрес с которого хотим просматривать содержимое памяти компьютера. Пока введите "STOP" (SYMBOL SHIFT+A) и "ENTER" и поговорим подробнее о самой программе-мониторе.

Здесь вам уже знакомы некоторые фрагменты: строки 8 и 9 - переключатели шрифтов (в программе проставлены в нужных местах GO SUB 8 и GO SUB 9). Для сохранения результатов Вашего труда Вы периодически будете делать RUN 5 - это запись на ленту программы-монитора и кодов MF 09 в том состоянии, до которого Вы дошли. После того, как сделаете RUN 5, программа запросит имя файла для записи. Можете задавать 1, 2, 3... и т.д.

Строки 10, 15, 20 - это вспомогательная подпрограмма, присваивающая значения переменным в соответствии с содержимым ячейки памяти. Строка 15 предохраняет от вывода на экран управляющих символов, имеющих коды с 0 по 31. Если встретится такой символ, то на экране будет инверсно напечатан знак "?". Так Вы сможете различать управляющий символ и настоящий знак вопроса, которые тоже будут встречаться в тексте.

Строка 210 зарезервирована для организации поиска нужной информации, к ней мы еще обратимся позже.

Строки 230 и 231 очень похожи. В строке 231 стоит REM и она, таким образом выключена. Строка 230 выдает на экран адрес ячейки памяти, код, содержащийся в ней и символ, соответствующий этому коду; далее - в кавычках - ожидается ввод символа, код которого надо записать в эту ячейку. Это удобно при замене текстовых сообщений, так как ввод осуществляется непосредственно буквой, на место того символа, который мы видим, измененный текст тут же отображается на экране. Удобно также и то, что переключившись на загружаемый символьный набор программы "MF09 R/L", мы будем видеть текст в том виде, каким он будет в готовой программе. В режиме курсора [L] - печатаем русские буквы, а в режиме [C] (CAPS LOCK) - латинские. Действуют также регистры "EXT. MODE" и "GRAPH".

Вводить новый текст надо по одной букве, не забывая после каждого символа нажимать "ENTER", а также ставить "пробелы" между словами. Если Вы ничего не хотите менять, то просто нажимайте "ENTER" для перехода к следующей ячейке. Для того чтобы, остановить программу, надо нажать "КУРСОР ВЛЕВО" (SYMBOL SHIFT+5), затем "STOP" и "ENTER". Потом опять RUN или GO TO 100 для работы с новыми адресами. Если же Вы увидели, что допустили ошибку при вводе, то остановите программу, сделайте GO TO 200 - программа стартует без запроса с того адреса, который Вы вводили последний раз. Нажимая "ENTER", подойдите к тому месту, где допущена ошибка.

Теперь поэкспериментируйте с подготовленными программами, а в следующем выпуске ZX-РЕВЮ мы доведём до конца рассказ о том, как выполнить полную русификацию программы MASTERFILE 09.

СОВЕТЫ ЭКСПЕРТОВ

STALINGRAD

CCS 1989 г.



Эксперт Захаров М. Ю. г. Казань

Введение.

Стратегическая игра "STALINGRAD", разработанная фирмой CCS в 1989 году, представляет дальнейшее развитие игры "OVERLORD", расширяя ее возможности. Таким образом, фирма создала модели двух важнейших сражений второй мировой войны, переломивших ее ход на Западе и Востоке. Однако, если в "OVERLORD" Вы управляли войсками союзников, то в "STALINGRAD" Вам предлагается управление немецкими войсками. Можно не упоминать лишний раз о "холодной войне" и "образе врага" - одним словом, нам, конечно, это досадно и единственное, что можно сделать - подождать, пока отечественные программисты напишут программы, адаптированные под наше понимание истории.

Сражения на советско-германском фронте, уникальные своим огромным пространством и большим напряжением сил, драматичностью, наверняка еще послужат базой для сюжетов новых стратегических игр (например сражения за Москву, Берлин, сражение у озера Балатон, форсирование Днепра, освобождение Крыма, оборона Ленинграда, операция "Багратион", наступление в Маньчжурии), реализующих и советскую сторону в войне.

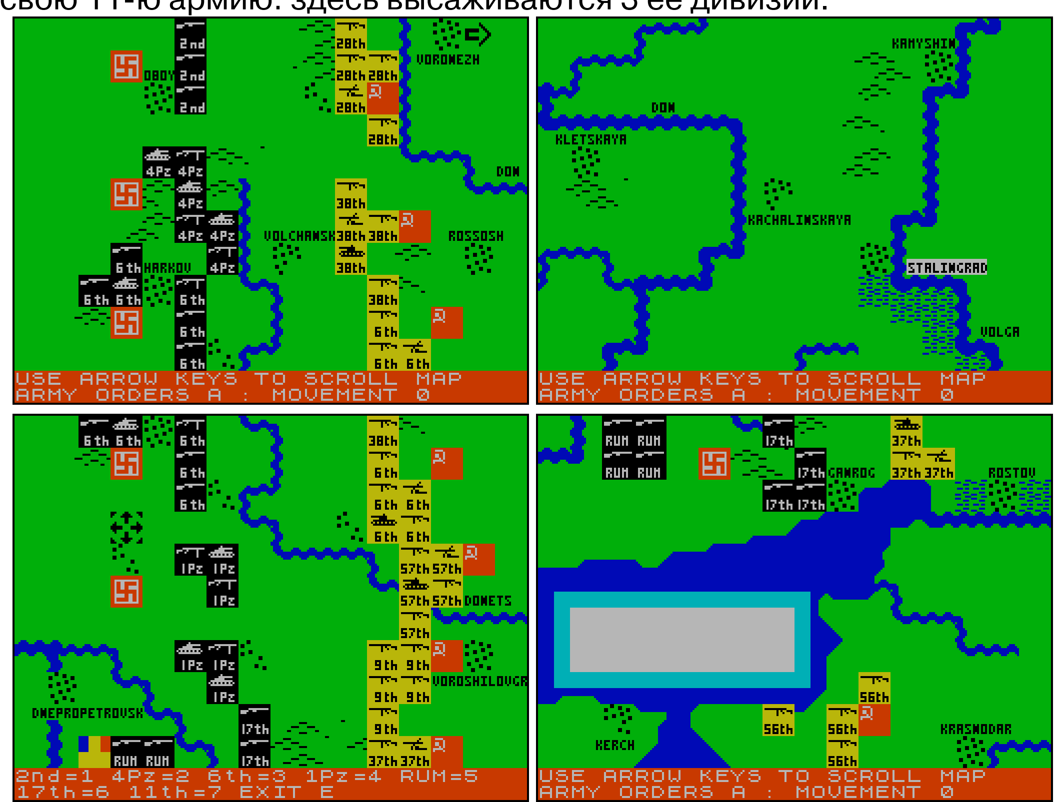
Вообще же, "STALINGRAD" - очень интересная игра, в ней реализованы такие возможности, как снабжение войск, корпусная организация армий, раздельное управление дивизиями (в OVERLORDе можно было управлять только армиями в целом). Боевые действия зависят от многих факторов (морального состояния войск, системы коммуникаций, рельефа местности и т. д.).

Она отражает события, происходившие на южном фланге советско-германского фронта с июня по декабрь 1942 года. В их результате Германия утратила, наконец, стратегическую инициативу, и перешла к обороне, как на Восточном, так и на Западном фронте.

К началу лета 1942 г. Германия была еще настолько сильна, что перешла в наступление на половине своего Восточного фронта. Цели этого наступления были весьма решительны: планировалось пересечь Кавказский хребет, завладеть нефтяными промыслами Азербайджана и нависнуть над английским Ближним и Средним Востоком, Ираном и Индией. Как сопутствующая цель рассматривался захват Сталинграда и перехват Волги, как важнейшей транспортной артерии, связывающей Советский Союз с предстоящим районом боевых действий, а заодно с бакинской нефтью и южными маршрутами поставок союзников. Красная Армия еще не обладала к этому моменту силами, достаточными для отражения такого наступления, хотя и пыталась его упредить. Попытки кончились для нас катастрофами под Харьковом и Керчью.

Наступая, немцы заняли Донбасс, затем перевалы Главного Кавказского хребта, Моздок, вышли к Сталинграду, но нараставшее сопротивление советских войск заставило Гитлера отложить свои далеко идущие планы, сосредоточившись на Сталинграде. Сталинград постепенно приковал все его ударные силы, и там они большей частью нашли себе могилу. Накопившая силы Советская Армия сама перешла в наступление, и 23 ноября вокруг Сталинградской группировки немцев замкнулось кольцо. Примерно этим периодом, судя по привлекаемым силам, завершается "STALINGRAD". Сталинградская битва же продолжалась. Советские войска отразили попытки немцев деблокировать окруженных, и к 2 февраля ликвидировали "котел".

Вы начинаете игру на позициях, примерно соответствующих линии фронта в июле 1942 г. В Вашем распоряжении 6 армий, одна из них в резерве - 3-я румынская (настроения в ней пониже, чем в немецких армиях). 11-я армия находится в Крыму. Вам противостоят слабо вооруженные и плохо оснащенные советские армии 1-го эшелона. Они пытаются перейти в наступление, но быстро терпят поражение. Этот этап вообще достаточно прост: Вы действуете вблизи баз снабжения, настроения в войсках высокие. Главная ударная сила в игре танковые дивизии. У Вас они сведены в 2 танковые армии, используйте их для быстрой победы. Попав хотя бы одной дивизией на Таманский полуостров, Вы активизируете свою 11-ю армию: здесь высаживаются 3 ее дивизии.



Однако кончается лето, начинается распутица, линии Ваших коммуникаций растягиваются. Вам приходится прикладывать все большие усилия для овладения новыми пунктами для баз снабжения. Численность ваших войск сократилась, а выделяемые верховным командованием пополнения скудны. Настроение в войсках падает. А тем временем из глубины Советского Союза подходят и разворачиваются свежие и хорошо оснащенные армии, имеющие большое количество танков и мотопехоты. (Обратите внимание на стрелки на краях карты военных действий: они обозначают полосы, в которых происходит это выдвижение), теперь все далеко не так просто!

Управление игрой

После загрузки программа запрашивает уровень сложности, предлагает загрузить отложенную партию. После этого Вы видите карту с расположением войск. Карту можно скроллировать клавишами курсора. В нижней части экрана находятся сменяющие друг друга меню.

Каждый ход состоит из 2-х этапов: отдания приказов и собственно перемещения войск. В начале каждого хода программа предлагает записать ситуацию (SAVE GAME - S),

отдать приказы (ARMY ORDERS - A) и перейти к их исполнению и перемещению войск (MOVEMENT - O).

Ваши армии состоят из 3-х родов войск: пехоты, мотопехоты, танков (каждый последующий род сильнее и маневреннее предыдущего). Аналогична структура советских войск, хотя, символы, обозначающие род войск различны.

Обратите внимание на символы с 4 стрелками. Это базы снабжения. Они могут располагаться лишь в населенных пунктах, являющихся узлами дорог. По мере продвижения Ваши поиски будут опираться на сеть этих баз. База появляется в очередном занятом Вами пункте, когда он оказывается у Вас в тылу.

Нажав в главном меню "A", Вы попадаете в меню принятия решений. Здесь Вы можете отдать приказы (ORDERS - O), просмотреть состояние сил обоих сторон (DETAIL - D) и их расположение (TERRAIN - T). Каждой армии соответствует цифровая клавиша, с помощью которой Вы к этой армии обращаетесь. Перейдя в режим ORDERS, Вы отдаете приказ выбранной армии, указывая ей рубеж, который она должна занять, и характер действий (наступление ATTACK - A, оборона DEFEND - D). Большинство Ваших армий, кроме 2-й и 11-й, включают по 6 дивизий, сведенных в 2 корпуса (3 дивизии в каждом), 2-я и 11-я армии состоят из одного корпуса каждая. Вы указываете последовательно положение центра, правого и левого флангов сначала одного, затем другого корпуса армии. Дивизии при этом помечаются символами "X", "R", "L" - то есть центр, правый и левый фланги корпуса. Таким образом, Вы можете указать желаемое место для каждой дивизии, что, по сравнению с OVERLORDом, большое удобство.

Прелесть же "самовольных" последующих действия дивизии, обусловленных принадлежностью ее к корпусу и армии, STALINGRAD унаследовал у OVERLORDa. Заданный рубеж может быть достаточно далеким: соединение будет выполнять свою задачу решая тактические вопросы с определенной долей самостоятельности. Однако если оно подверглось удару противника и стало откатываться назад, приказ (если Вы не отменяете его) лучше продублировать.

В режиме "DETAIL" Вы можете просмотреть численность войск обоих сторон. Вы увидите и настроение своих частей:

EXLT - excellent - превосходно
V. G. - very good - очень хорошее
GOOD - хорошее
FAIR - благоприятное
POOR - ухудшенное
LOW - низкое
ABYS - отчаяние

От настроения зависит эффективность ваших войск, их потери. В режиме "TERRAIN" программа выделит расположение выбранной вами армии. Ее дивизии закрашиваются одноцветными квадратиками без всяких надписей, что доставляет большое неудобство, поэтому этим режимом лучше не пользоваться. Выбравшись из младших меню в главное посредством многократных нажатии "E" (Exit), Вы и можете привести войска в движение, нажав "O" (MOVEMENT). Программа поочередно показывает движение и нанесение ударов обеими сторонами (удары наносят, разумеется, только наступавшие части). Вы видите потери, которые несут сражающиеся дивизии в процентах. Между действиями немецких и советских войск Вам предлагается пополнить свои части. В рамке-меню, где-то в районе Азовского моря, перемещая указатель клавишами "P" и "Q", Вы выбираете очередную пополняемую армию, нажимая "A". Если армия отрезана от баз снабжения, доступа к ней нет. Затем - род войск пополнения (фиксация - "T") и его численность (SET VALUE - V). В этом меню Вы видите размеры имеющихся у Вас резервов:

ARM - танки
MECH - мотопехота
INF - пехота (инфантерия)

На протяжении нескольких первых ходов пополнения Вам не выделяются.

В ходе боевых действии некоторые дивизии не просто несут потери, но исчезают совсем: они могут быть расформированы (Unit disband) или разгромлены (Unit routs). Остатки расформированной дивизии пополняют другие дивизии армии. Вместо исчезнувших дивизий можно сформировать новые из пополнения любого рода войск. Таким образом, в состав пехотных армий можно включить танковые и мотопехотные дивизии, и наоборот (но не стоит распылять силы).

Формирование новой дивизии произойдет, если передать армии большое пополнение - 100 или 200.

Армия не может превзойти своих первоначальных размеров: она не может включать больше 6 дивизий, а дивизия не может иметь более, чем 100% состав.

Когда все Ваши резервы будут исчерпаны, программа выдаст Вам свое заключение о том, кто же победил, об устойчивости победы и сообщит количество выживших в результате Ваших действий у обеих сторон. Заключение вполне может не оправдаться, Вы можете продолжить игру, ответив "Y" на соответствующий вопрос. Правда, игра с каждым ходом будет все больше напоминать эндшпиль шахматной партии, когда на доске остается по 2-3 фигуры.

Некоторые детали

Пусть Вас не удивляет, что последние советские армии первого эшелона стремительно расформировываются, выйдя из соприкосновения с Вами. Они очищают место для вступления на сцену II эшелона, вливаясь в его дивизии.

Обратите внимание на то, что программа перемещает армии последовательно, по их расположению с севера на юг вдоль линии фронта. Поскольку порядки одной армии плохо проницаемы для другой, это правило стоит учитывать, если Вы не хотите, чтобы Ваши войска перепутались.

Компьютер скрывает расположение своих частей, не находящихся в контакте с Вашими. Но, если Вы вызовете данные о них в режиме "DETAIL" или "TERRAIN", он проскроллирует карту и укажет приблизительный район их расположения, хотя, конечно, цветом их не выделит.

Следите за целостностью своих коммуникаций. Снабжение армии может быть прервано по нескольким причинам: она далеко оторвалась от баз снабжения, советские войска совершили рейд по Вашим тылам (нередко компьютер направляет Вам в тыл танковые бригады), наконец, Вы сдали свою базу, отходя. Лишившись снабжения, армия сразу теряет в эффективности: падают маневренность, настроение в войсках, растут потери и их нечем восполнять. Для разгрома такой армии достаточно нескольких ходов. Чтобы восстановить снабжение, надо заново создать всю систему коммуникаций, для чего армия должна вплотную подойти к любой базе снабжения. Нередко приходятся довольно долго отступать.

Уровни сложности игры

В начале игры программа предлагает выбрать уровень сложности (ENTER GAME LEVEL). Их три: (1-3).

Принципиальных различий между ними нет, но из-за многочисленных мелких сложностей игра на 3 уровне значительно труднее.

Выгрузка файлов на ленту

Игра достаточно продолжительна, полный ее цикл длится около 8 часов, поэтому Вам наверняка придется записывать партию на ленту, и не один раз. Программа предлагает сделать это в начале каждого хода. Перед выдачей информации на магнитофон выдается предупреждение (Press any key). Записываемый файл имеет длину более 16 килобайт, в нем сохраняются не только положение войск, но и отданные Вами приказы.

Должен предупредить об одной детали. Бывает, что программа выводит сообщение "Unit disband" уже после записи партии. В этом случае на моем компьютере версии "Балтика" записанный файл оказывается сбойным. После его загрузки игра "виснет".

Приходится дожидаться следующего хода и записывать партию заново.

Программа предлагает загрузить отложенную партию только один раз, в начале игры, после ввода уровня сложности.

STALINGRAD принадлежит к тем немногим играм, которые не могут быть запущены заново без перезагрузки из-за большого объема информации, характеризующей ситуацию. Поэтому нельзя прервать неудачную партию иначе, как только перезагрузив программу. По достижении победы одной из сторон программа тоже предлагает считать себя заново с ленты.

Замечания

События, реализуемые игрой, начальное положение сторон, состав сил, стартовые действия, завязывающие ситуацию, конечно, лишь напоминают реальные события 50-летней давности. Все мои попытки повторить Сталинградскую битву не привели к успеху. Советские войска в игре не включают ни 5-й танковой армии, ни танковых и механизированных корпусов, танки и мотопехота рассеяны по общевойсковой армиям, 2-я немецкая армия в действительности была венгерской. 11-я армия вполне самостоятельно переправилась из Крыма летом 1942 г. Существует и много других немаловажных деталей, отличающих Сталинград от STALINGRADа.

Но, как самостоятельная игра, STALINGRAD очень интересна и наверняка привлечет ваше внимание.

FLIGHT SIMULATION[1]

Psion 1983 г.

Перевод фирменной инструкции "ИНФОРКОМ", 1987

В последние годы пилотов все чаще обучают, используя имитаторы. Даже на маленьком компьютере можно реально, в реальном времени, отобразить многие особенности полета: динамику самолета, навигацию, инструментальное обеспечение и т. д. "Имитатор полета" включает в себя эти эффекты и представляет модель полета маленького двухмоторного винтового самолета.

Аспекты полета

Органы управления самолетом включают в себя: штурвал, закрылки, хвостовой руль и регулятор мощности двигателей. Движение штурвала вперед и назад вызывает отклонение горизонтальных рулей и, соответственно, перемещение самолета вниз или вверх.

Аэродинамика самолета чрезвычайно сложна. Управление каким-либо органом вызывает, как правило, не одно последствие. Например, элероны не только вызывают крен самолета, но и создают дополнительные боковой воздушный поток, вызывавший поворот самолета. Всему этому можно научиться, работая с имитатором.

Положение самолета и характер полета отображаются на приборной панели в кабине

[1] Последующие версии носят название Flight Simulator (Прим. NUK)

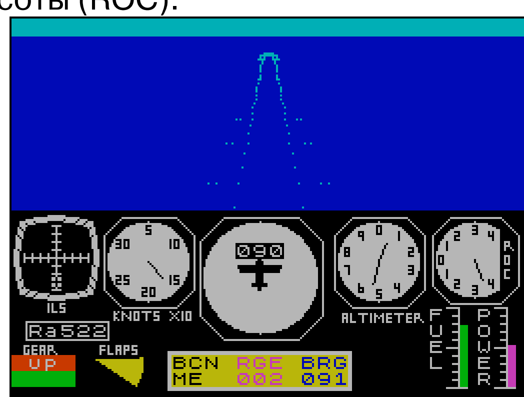
пилота. Пилот должен руководствоваться показаниями этих приборов при выведении самолета на линию, с которой он совершит посадку с требуемой скоростью, под необходимым углом. Обычно правильный угол посадки составляет 3 град., т.е. высота должна быть 6000 футов на расстоянии 20 миль, 3000 футов на расстоянии 10 миль и 1000 футов на расстоянии 3 мили от ВПП. Руль направления также может несколько изменять курс самолета. При движении по ВПП именно им регулируется направление движения самолета.

В имитаторе Вы можете приземляться на одном из двух аэродромов, взлетать с них, ориентироваться с помощью радиомаяков или по местности. Через кабину Вы видите светлое небо и темную землю, огни взлетно-посадочной полосы в трехмерной перспективе и объекты на земле: озеро и т.п. Экран можно переключить также на изображение навигационной карты, на которой показаны детали местности: ВПП, радиомаяки и пр. После загрузки программы в машину на экране появляется меню с вопросом: "Что Вы хотите отрабатывать?" - взлет, гладкий полет или посадку. Нажмите, соответственно, 1, 2 или 3. После этого Вас спросят, нужен ли вам учет ветра. Отвечайте "Y" только если Вы опытный пилот и можете справиться с внезапными порывами ветра, в противном случае - "N".

Приборная панель

На нижней части экрана изображена приборная панель кабины пилота. Пять циферблатов слева-направо это:

1. Система инструментальной посадки (ILS).
2. Датчик скорости.
3. Оборудование наведения на радиомаяки (RDF).
4. Высотомер.
5. Индикатор скорости высоты (ROC).



RDF - это большой циферблат в центре панели. Здесь изображен символ самолета, он указывает направление полета самолета. Цифровой указатель дает направление полета в градусах компаса. RDF - это самая необходимая навигационная система, в любое время полета самолет связан с одним из радиомаяков. Положение маяка, к которому в данный момент подключился самолет, изображается светлым крестиком на окружности циферблата RDF. Если Вы хотите двигаться на данный маяк, Вам надо развернуть самолет так, чтобы положение маяка совпадало с "12 часами" на циферблате RDF.

Датчик скорости находится справа от RDF. Малая стрелка указывает высоту в тысячах футов, а большая в сотнях футов.

Индикатор ROC - это индикатор скорости подъема. Он измеряет вертикальную скорость самолета в единицах 1000 футов/мин. Когда стрелка выше нуля, самолет идет вверх и наоборот.

Индикатор POWER - "мощность" - измеряет тягу моторов, тяга двигателей возрастает при увеличении POWER, но на большой высоте падает.

FUEL - "топливо" - наличие топлива в баке.

FLAPS - "закрылки" - показывает угол выдвижения закрылков.

GEAR - "шасси" - имеет два состояния. Шасси убрано - красный цвет; шасси выпущено - зеленый цвет.

BCN, RGE, BRG - информация по маяку, к которому привязан в данный момент самолет. BCN - позывной маяка. RGE - расстояние до маяка в милях. BRG - курс в градусах относительно маяка. ILS -система инструментальной посадки. Эта система для наведения самолета на аэродром. Радиобуй, размешенный в начале ВПП, излучает сигнал, положение которого видно на экране ILS, как светящееся пятно. Когда самолет приближается к аэродрому с правильного курса, светящееся пятно будет в центре экрана.

Ra - радиовысотомер, это часть системы ILS. Радиосигнал, отраженный от земли, позволяет замерить высоту от земли до колес самолета. Замер в футах. Это точный отсчет; применяется при посадке.

Управление самолетом

Управление влево-вправо-вверх-вниз с помощью курсора - клавиши 5, 6, 7, 8.

Управление горизонтальным рулем - Z - поворот влево. X -вправо. Им же управляется самолет при движении со ВПП.

Тяга двигателя: P – повышение, O - понижение.

Закрылки:

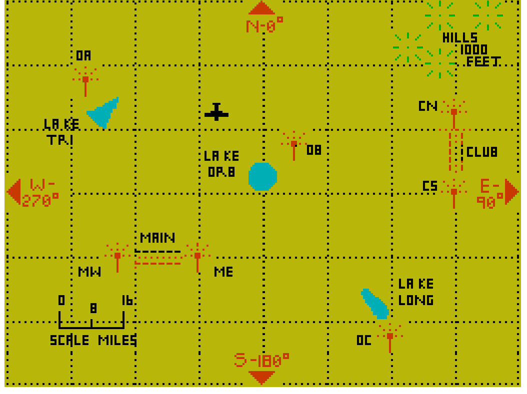
F - увеличение выхода закрылков; D - втягивание закрылков.

С убранными закрылками скорость отрыва самолета - 80 узлов, с выпущенными закрылками - 60 узлов. Увеличение выхода закрылков при большой скорости полета может привести к разрушению крыльев.

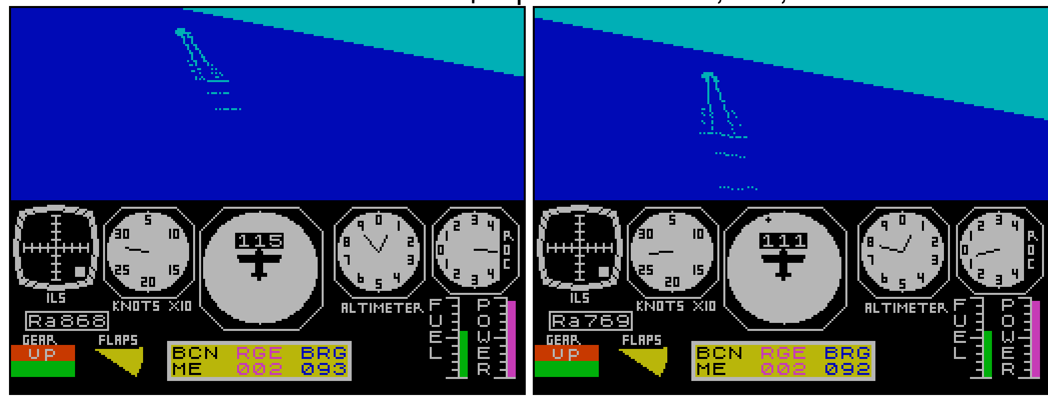
Шасси: выпуск и втягивание - клавишей G. Нельзя выпускать шасси при большой скорости, т.к. оно может заклинить или оторваться.

"Радиомаяки": - чтобы подключиться к другому маяку, нажмите клавишу B. Пока Вы будете держать эту клавишу, изменяется маяк, к которому привязан самолет.

Карта: - нажав кнопку M, Вы получите на экране изображение навигационной карты. На карте изображены два аэродрома: один международный аэропорт, MAIN и один маленький клубный аэродром CLUB. MAIN имеет длинную ВПП длиной более мили и поэтому посадка на него для небольшого самолета несложна, CLUB - это маленький местный аэродром и имеет полосу 800 ярдов.



Карта изображает также положение маяков и различных наземных объектов. Около главного аэропорта есть два маяка, расположенные на расстоянии трех миль от начала и от конца полосы. Их позывные ME и MW. Есть еще три маяка: OA, OB, OC.



Навигация

Самая трудная часть полета это подход к аэродрому и посадка. На достаточно большой высоте Вы можете экспериментировать с органами управления, меняя скорость и направление полета, не заботясь о навигации. Если же Вы хотите совершить посадку, Вам надо положить самолет на правильный курс и подойти к аэродрому с правильным углом посадки. Это трудная задача и она требует больших усилий и тренировки, пока Вы совершите посадку правильно.

STAR RAIDERS

"Звездные рейнджеры"
ATARI 1987



Эксперт Порядин С. В.
Марийская ССР

Игра STAR RAIDERS 2, разработанная фирмой ATARI, органично объединяет в себе несколько игровых жанров. В этой игре Вы можете проявить себя, как стратег в планировании операции против захватчиков из другой звездной системы, так и как пилот боевого космического корабля в схватках с эскадрами кораблей захватчиков.

Боевые действия ведутся как в открытом космосе в двух звездных системах: Вашей "CELOS" и вражеской "PROCYON", так и над поверхностями планет, на которые высадились вражеские корабли. В Вашей звездной системе вокруг центрального светила вращаются три планеты, одна из которых имеет спутник. Все планеты и спутник заселены и основное население сконцентрировано в городах. Уничтожение городов является главной целью захватчиков.

В Вашей звездной системе также находятся несколько космических станций, на которых отремонтируют Ваш корабль, дозаправят его горючим и боеприпасами.

Враги

Они прилетают из своей звездной системы. Их корабли никогда не появляются в одиночку, они все время летают эскадрами, в состав эскадры входит несколько флайеров (иногда их число больше дюжины), два или три крейсера и один флагманский корабль.

Флайер имеет легкое вооружение и может быть сбит единственным попаданием, но флайеры нападают сразу вдвоем или втроем, так что неопытному пилоту будет поначалу очень "жарко". При уничтожении всех флайеров в эскадре противника наступает черед мощных и опасных крейсеров. Они имеют очень мощное оружие и могут быть поражены одним или несколькими попаданиями бомб. Смена оружия в вашем корабле происходит автоматически, как только на Вас нападет крейсер или флагманский корабль. Но космические битвы - это не то, для чего предназначены крейсера (хотя они очень маневренны). Основная их цель уничтожение городов на Ваших планетах.

Флагманский корабль гораздо мощнее вооружен чем крейсер, но опасности для Ваших планет он не несет. Правда, за уничтожение флагмана Вам будет начислена большая сумма очков. Но основная Ваша цель - это защитить города на планетах.

Вражеские эскадры также могут атаковать станции. Если врагов вовремя не отогнать

от них, то станции могут быть уничтожены и Вы останетесь без баз. На нижнем уровне игры, когда у Вас имеется три станции, потеря одной из них не несет больших неудобств. Но на более высоком уровне это может обернуться для Вас катастрофой.

Уничтожая вражеские эскадры, невозможно добиться победы, так как из вражеской звездной системы в Вашу будут лететь все новые и новые экспедиции. Чтобы одержать полную победу над врагом, нужно уничтожить его базы. Базы врага находятся на трех планетах его звездной системы. Уничтожить базу противника можно метким попаданием атомной бомбы. Для включения системы бомбометания при полете над поверхностью планеты необходимо нажать клавишу "W". Тогда внизу обзорного экрана появится прицел для бомбометания. При приближении к базе на медленной скорости нужно вывести ее изображение под прицел и нажать "огонь". За один заход невозможно уничтожить все базы противника, т.к. у Вас ограничен запас бомб, так что придется возвращаться на станцию для пополнения боезапаса.

Запуск игры

После загрузки программа предложит вам выбрать тип джойстика и игровой уровень от "1" до "3". После нажатия "S. SHIFT" Вы приступаете непосредственно к игре.

После запуска игры Ваш корабль находится на одной из планет своей звездной системы. Он сразу же будет атакован вражескими флайерами, но это лишь разведчики врага, и никакой опасности для городов они не представляют. Поэтому Вы можете сразу отлететь от планеты, нажав "SPACE", выбрав курс и затем нажав клавишу "огонь". Теперь, в более спокойной обстановке, можете внимательно рассмотреть находящееся на экране Вашего дисплея изображение.

Экран

Весь экран можно условно разделить на две части. В верхней, меньшей его части, расположена приборная доска вашего корабля. На ней находятся несколько индикаторов, показывающих состояние его систем.



В левой верхней части расположен индикатор энергии, чуть ниже этого индикатора расположено два ряда прямоугольных меток - это счетчик атомных бомб, имеющихся в Вашем распоряжении. Ниже этого счетчика находятся три индикатора, показывающих, какое оружие в данное время Вами задействовано.

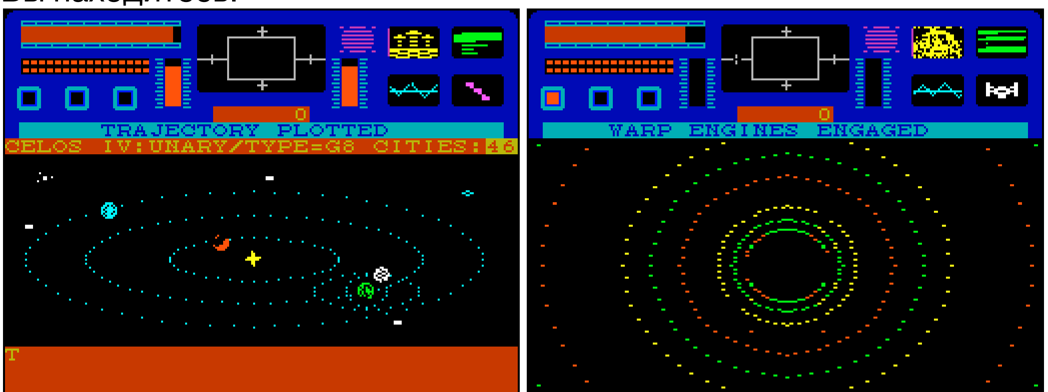
В центре приборной доски находится экран радара, который позволит ориентироваться в сложной обстановке боя. Если нажать клавишу "T", то изображение на нем сменится изображением Вашего корабля, на котором условными метками будут указаны системы, пострадавшие от вражеского огня. Если контур корабля на экране радара обнесен точками, это значит, что включено защитное поле, в местах, где поле пробито огнем вражеских кораблей точки отсутствуют или мигают.

По обе стороны экрана радара находятся две вертикальные шкалы показывающие температуру лазеров, ниже расположен счетчик очков, заработанных Вами.

Остальная информация на приборной доске не несет большой смысловой нагрузки.

Ниже приборной доски расположен экран Вашего корабля, на котором и показывается обстановка по курсу. Между экраном и приборной панелью находится информационная строка, на ней в ходе игры будут появляться сообщения о различных действиях врага и о критическом уровне энергии в Вашем корабле.

Если во время игры нажать клавишу "SPACE", то на экран корабля будет спроецирована карта звездной системы, в которой он находится. При этом внизу экрана расположена информация о планете, на которую Вы направили указатель гиперперехода или на который Вы находитесь.



Управление

Управление кораблем не очень сложное и осуществляется с помощью выбранного джойстика. При полете над поверхностью планеты движением ручки джойстика вверх-вниз регулируется скорость Вашего корабля.

Клавиша "S" служит для вкл/выкл защитного поля.

Клавиша "W" служит для переключения стрельбы лазером на бомбометание и обратно. На то, какое оружие используется, указывает соответствующий индикатор.

Клавиша "T" служит для переключения экрана радара на индицирование состояния систем корабля. При повторном нажатии включает радар.

Клавиша "P" служит для временного останова игры.

Клавиша "C. SHIFT" заканчивает игру и выводит на экран заставку.

Для гиперперехода необходимо вывести на экран карту звездной системы, в которой Вы находитесь, и, манипулируя джойстиком, нужно установить указатель гиперперехода на любой объект в этой системе. Если после этого нажать "огонь", не выходя из "карты", то переход будет сделан туда, куда Вы указали.

Перелет в другую звездную систему можно осуществить, предварительно установив указатель гиперперехода на ее символьное изображение в углу карты и нажав клавишу "огонь".

Энергия

Энергия расходуется на поддержание защитного поля во время обстрела Вашего корабля вражеским и во время гиперпереходов. При полном расходе запаса энергии корабль погибает.

Теперь несколько советов, которые могут быть Вам полезны.

Старайтесь не допустить высадки вражеской эскадры на планету в Вашей системе, так как сразу после этого крейсера начнут уничтожать города, да и сбить крейсер над планетой значительно сложнее.

Не старайтесь полностью уничтожать эскадру в космосе. Достаточно уничтожить все крейсера. Флагманский корабль, оказавшись в одиночестве, не представляет серьезной угрозы, а схватка с таким мощным кораблем после боя с крейсерами, когда в Вашем корабле мало энергии, может оказаться роковой.

При нападении на станцию вражеской эскадры, ее изображение на карте начинает мигать и Вам нужно спешить к ней на выручку, но если у Вас есть более срочные дела, например уничтожение высадившихся на планету врагов, то разборку с врагами, напавшими на станцию, можно отложить. Некоторое время станция может продержаться сама. Однако, следует помнить, что на более высоких уровнях игры это время значительно сокращается, да и станций там меньше (на третьем уровне - всего одна станция). Так что рисковать не советуем.

Из этой войны Вы выйдете победителем, если уничтожите все базы противника и все

эскадры вражеских кораблей, но при этом у Вас должен остаться хотя бы один город на любой из планет. Если же враги уничтожат все Ваши города, то это будет поражение. Естественно, Вам будет засчитано поражение, если враги смогут сбить Ваш корабль. На первых порах это и будет происходить чаще всего, но, потренировавшись и накопив боевой опыт, Вы станете серьезным препятствием на пути космических агрессоров.

THE TRAIN

("Поезд")
Accolade 1988.
Эксперт Порядин С. В. Марийская ССР

The Train представляет собой очень оригинальную программу-иммитатор. В этой игре Вам будет предложено испытать свои силы в качестве машиниста паровоза. Кроме того, в ней присутствуют элементы других игровых жанров, в том числе и стратегических игр.

Краткий сюжет

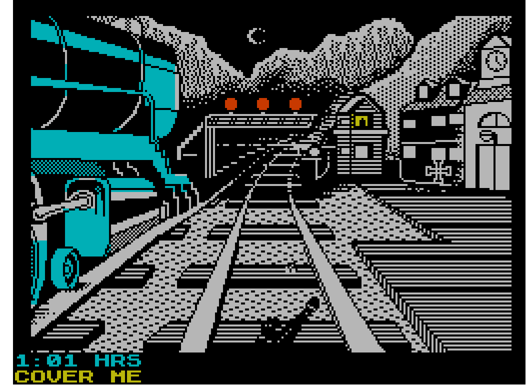
Представьте себе, что Вы находитесь в центральной части Франции в 1944 году. Войска союзников высадились на западном побережье и ведут бои с войсками вермахта. Вы командир одного из отрядов сопротивления, и Вашему отряду дано задание отбить на станции METZ (г. Мец) бронепоезд и провести его к войскам союзников.

От Вашего умения ориентироваться в сложной обстановке зависит успех данного предприятия. Вам нужно спланировать маршрут движения бронепоезда по железным дорогам Франции. По пути придется прорываться через станции и железнодорожные мосты, захваченные фашистами. Их нужно отбить у врага. В пути на бронепоезд будут совершать налеты самолеты Люфтваффе и Вам придется от них отбиваться. Ну и кроме всего этого, от Вас потребуется непростое умение справиться с такой сложной машиной как паровоз. Ведь кроме опасностей, подстерегающих Вас на пути, могут произойти большие неприятности связанные с неумением справиться с органами управления в кабине паровоза. Вы можете погибнуть от взрыва котла или, наоборот, бронепоезд остановится из-за нехватки давления пара, если Вы неправильно рассчитаете маршрут движения и у Вас кончится уголь или вода. У паровоза также могут выйти из строя тормоза, и игра на этом закончится.

Настройка программы

После запуска программа предложит выбрать джойстик и сразу после этого Вы приступите непосредственно к игре.

В начале игры Вы находитесь с пулеметом на путях возле паровоза. Ваша задача - прикрыть своего помощника Ле Дюка, который направился к стрелке, чтобы перевести ее. Вам нужно уничтожать немецких солдат, силуэты которых будут появляться в окнах зданий расположенных поблизости. Запас патронов у Вас неограниченный и можно порекомендовать не отпускать гашетку и стрелять длинными очередями.

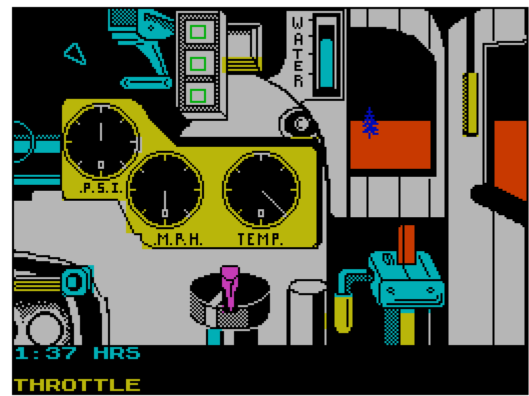


Как только Ле Дюк доберется до семафора, Вам будет предложено выбрать путь, по которому пойдет бронепоезд. Путь выбирается с помощью клавиш управления или джойстика. Затем нужно прикрыть возвращение Вашего помощника. Как только он

доберется до паровоза, Вы автоматически оказываетесь на рабочем месте машиниста и приступаете к основной части игры.

Экран

Перед Вами расположено несколько приборов и органов управления паровозом. На приборах показывается: скорость паровоза (MPH), давление пара (PSI), его температура (TEMP) и уровень воды (WATER).



К органам управления относятся: тормоза (BREAKE), рычаг реверса (FORWARD/REVERSE LEVER), заслонка сброса пара (STEAM BLOWOFF), клапан регулировки давления (THROTTLE), кроме этого перед Вами в кабине находятся крышка топки и ручка свистка.

Управление

Управление производится с помощью джойстика и некоторых клавиш на клавиатуре.

В кабине паровоза Вы можете перемещать с помощью джойстика стрелку, указывающую на какой-либо орган управления, с помощью джойстика можно и манипулировать им.

Чтобы поднять температуру и давление пара, необходимо открыть крышку топки и подбросить туда уголь (рукоятка джойстика вправо). Скорость движения бронепоезда регулируется рычагом управления давлением пара и тормозами. Чем больше Вы потянете на себя рычаг регулировки давления, тем больше пара будет поступать в цилиндры и тем быстрее будет двигаться бронепоезд. Для остановки нужно перекрыть поступление пара в цилиндры, толкнув рычаг до упора от себя и затормозить, повернув рукоятку тормоза. Приводить тормоз в действие лучше кратковременными импульсами по 2-3 секунды, иначе он быстро выйдет из строя, при закрытой заслонке начинает расти давление пара в котлах и, чтобы удержать его в норме, необходимо стравливать пар с помощью заслонки (STEAM BLOWOFF).

Из кабины паровоза Вы можете перейти к одному из зенитных пулеметов, расположенных спереди и сзади бронепоезда. Это необходимо при отражении атак немецких самолетов. Переключение осуществляется нажатием клавиш "1" или "2": возврат в кабину - "3". О налете вражеской авиации Вас проинформирует компьютер, он же выдаст сообщение, если давление пара будет превышать норму или будет слишком низким. Компьютер проинформирует Вас и в том случае, когда запасы угля или воды будут подходить к концу.

Включив "Таблицу состояния" с помощью клавиши "5", можно узнать о состоянии различных систем бронепоезда, из таблицы можно узнать о процентном износе тормозов, котла, брони, и также о количестве оставшегося угля. Выход из таблицы - нажатием клавиши "5".

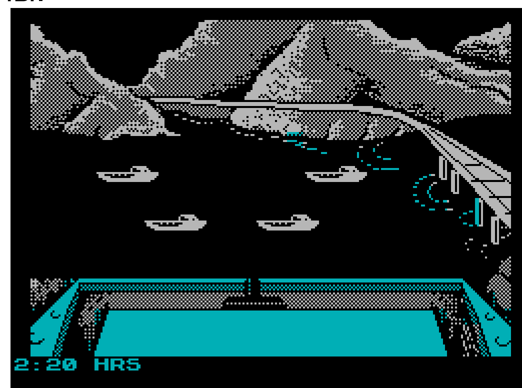
С помощью клавиши "4" можно вывести на экран карту сети железнодорожных путей запада Франции, на которой будут отмечены все станции, мосты, развилки и также указано положение Вашего бронепоезда. По этой карте Вы узнаете о приближении бронепоезда к станции или мосту, а также спланируете дальнейший путь состава по оптимальному маршруту. Другие клавиши:

"SPACE" - пауза.

"R" - выход из игры.
"S" - вкл/вкл звука.

Мост (BRIDGE)

На мосту, захваченном союзниками или отрядами сопротивления (о чем свидетельствует изменение его цвета на карте), Вы можете не останавливаться. Но как только попытаетесь проехать через мост, который контролируется немцами, без остановки, то будете немедленно уничтожены.



Поэтому при подходе к мосту необходимо замедлить скорость до минимума и, когда расстояние до моста сократиться до нуля, о чем Вас проинформирует компьютер, полностью остановить бронепоезд. В этот момент Вы автоматически окажетесь в орудийной башне и должны будете расстрелять немецкие речные суда, которые тоже будут отвечать огнем. Как только все суда будут потоплены, путь станет свободен и можно двигаться дальше.

Станция (STATION)

На станциях, захваченных врагом, можно не останавливаться, но после особенно длинных перегонов или перед ними необходимо заправить баки водой и загрузить уголь, это можно сделать лишь отбив у врага станцию. Для захвата станции нужно действовать аналогично случаю с мостом. После остановки Вы окажетесь на месте пулеметчика и Ваша задача - подавить сопротивление немцев. После захвата станции Вам предложат ознакомиться со сводкой германского командования. Нажав клавишу "Огонь", Вы увидите следующее меню:

- захват следующей станции;
- захватить следующий мост;
- сделать ремонт;
- ничего не делать.

Вы можете отправить соответствующее послание союзникам и получите ответ, в котором будет указано, в какое время произойдет данное событие (текущее игровое время указывается на циферблате в углу экрана).

После этого Вы можете продолжать движение.

Развилка или перекресток (SWITCH)

Повернуть на другие пути можно только при полной остановке бронепоезда на развилке при помощи рычага реверса.

Внимание!

Не трогайте рычаг до полной остановки бронепоезда.

На первых порах будет очень трудно справиться с управлением бронепоездом, но потренировавшись некоторое время, можно приобрести опыт и добиться больших успехов в этой необычной компьютерной игре.

DEATH WISH 3

("Жажда Смерти - 3")
Gremlin Graphics 1988 г.



Эксперт Троекуров В. И. г. Киев.

Фирма GREMLIN GRAPHICS выпустила свою программу на рынок вслед за появлением третей серии всемирно известного кинофильма "Жажда Смерти", главную роль в которой играет очень симпатичный и любимый многими мужественный Чарли Бронсон.

Те, кто не знаком с этим киносериалом, нередко полагают, что где-то существуют программы DEATH WISH 1 и DEATH WISH 2. Но к сожалению это не так. Цифру "три" программа получила от третьей серии фильма.

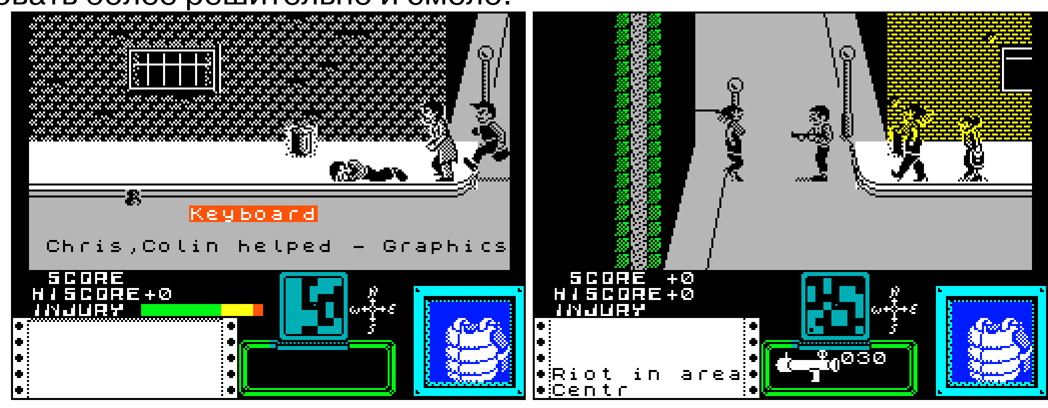
Краткое содержание картины. Скромный, хотя и талантливый архитектор, враг всякого насилия, дрожащими руками берется за оружие в первой серии картины, когда грязные бандиты убивают его жену и тяжко травмируют дочь. Выйдя в одиночку на ночные улицы города, он начинает свою большую и страшную месть всем, кто не дает людям спокойно жить. Постепенно в его действиях начинает проявляться профессионализм. Пролив реки крови, он остается на свободе, поскольку полиция, мягко говоря, смотрит сквозь пальцы на его подвиги - во всяком случае работы ей становится с каждым днем все меньше и меньше.

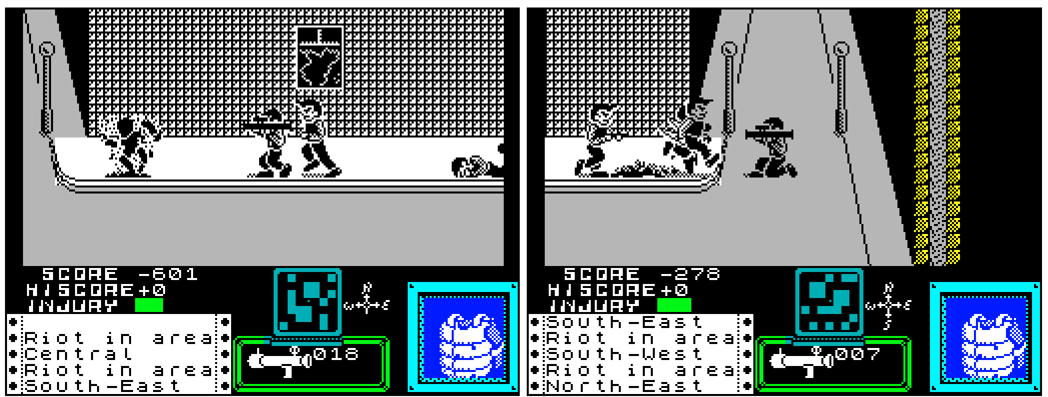
К третьей серии картины он уже сделал несколько безуспешных попыток "завязать", но обстоятельства по-прежнему вынуждают его снова и снова браться за оружие, Теперь он уже большой профессионал с известным по всей стране прозвищем "Мститель". Несколько разрушенных до основания кварталов и полное очищение города от неуправляемых банд - достижение третьей, но не последней серии картины.

Игра относится к жанру ACTION, но поскольку по ходу игры необходимо собирать оружие, и есть возможность произвольного выбора маршрута, следовательно в ней есть элементы ARCADE/ADVENTURE.

Сюжет: полиция не может освободить город от бандитов, ведущих между собой борьбу за власть в городе. Вооруженные банды устраивают стычки с полицией, во время которых проливается кровь мирных жителей.

Вы в качестве главного героя должны навести порядок в городе, а так как Вы свободны и независимы, в отличие от полиции, которая связана законами и уставами, то можете действовать более решительно и смело.





После загрузки программы нажмите "BREAK". Вы увидите нашего главного героя с винтовкой - это Стив, которым Вы управляете. Теперь можно начинать игру.

В нижней части экрана выводятся:

- выбранный вид оружия и боезапас;
- информационное табло, где приблизительно указывается местонахождение главарей банд (их пятеро и находятся они в пяти районах города: NORTH - WEST, NORTH - EAST, SOUTH - WEST, SOUTH - EAST, CENTRAL);
- состояние бронежилета (повреждения от выстрелов, возможность замены на новый);
- карта участка города показывает ваше местоположение (с указанием направлений: север, юг, запад, восток. Синий цвет карты - Вы настроены на поиск оружия, желтый - поиск главарей банд);
- SCORE - заработанные Вами очки;
- HI SCORE - высший результат предыдущих игр;
- INJURY - состояние Вашего здоровья (если показалась зеленая полоса, спрячьтесь в укромном месте и отдыхайте до тех пор пока полоска не исчезнет);

Итак начинайте игру. На улицах и в домах на Вас нападают бандиты, вооруженные винтовками или дубинками. Они не церемонятся - нападают и убивают на месте. Иногда будут пробегать полицейские и стрелять по бандитам, но от них мало пользы и рассчитывать Вам надо только на свои силы.

Оружие. У Вас имеется:

- пистолет;
- автомат;
- карабин;
- бронежилет.

Оружие и бронежилеты находятся в разных частях города - в домах, куда Вы можете зайти и пополнить боезапас или сменить вышедший из строя бронежилет.

Смена оружия производится нажатием клавиши "C". Нажатие клавиши "M" поможет Вам найти главарей банд, повторное нажатие "M" поможет найти оружие, боеприпасы, бронежилет. Пауза - клавиша "H".

Главари банд также находятся в домах. Для входа в дверь служит клавиша ENTER. Когда Вы найдете главаря, его надо расстрелять. Первым выстрелом его не убить, но зрелище того, как рассыпается стол, за которым он сидит, доставит Вам большое удовольствие.

Крайне нежелательно стрелять в простых мирных граждан - за это Вас штрафуют, но самое большее преступление, которое Вы можете совершить - убийство полицейского. За ним последует солидный штраф в очках и необходимость спасаться не только от бандитов, но и от полиции, а это уже конец.

FORUM

Проблемы ELITE

Мы по-прежнему получаем сотни писем по программе ELITE, но сейчас информация, содержащаяся в них, как правило повторяется. Какие темы в основном волнуют читателей:

1. Беспричинные захваты корабля на пиратских станциях. Очень много жалоб по этому поводу. После приземления сообщают, что Ваш корабль захвачен пиратами и действуют они безжалостно. Вам приходится начинать игру сначала. В качестве конкретного адреса приведем сообщение Тихомирова В. В. и Митрохина В. А. из г. Черноморска (Крым) о том, что так ведет себя станция на планете LAZASO в галактике N5.

2. По-прежнему много сообщений о чудачествах в версии, взломанной Родионовым. Здесь и галактика №47 и галактика № 1 без планеты LAVE и головокружительные суммы кредов и необъятные возможности вооружения - в общем, не знаем, хочется ли Вам в этот бред играть, но писать о нем нам не хочется.

3. После того, как мы в 10-м номере прошлого года опубликовали изображения всех документированных кораблей (как оказалось не зря!) пилоты начали настоящую охоту за НЛО. Сообщений много и как правило они имеют одиночный характер, но в десятках писем сообщается одно и то же. Есть корабль больших размеров, бесформенный, похожий на астероид, несущий на себе "Крейты" и "Саидуиндеры". Сам первым не нападает и, выпустив истребители, старается уйти от боя. Если его поразить, сбрасывает контейнеры. Факт существования такого недокументированного корабля можно считать установленным точно. Его поведение описывают десятки людей одними и теми же словами. Это кандидат на то, чтобы считаться "Космической платформой". Условно назовем его пока STRANGER ("странник").

Рисунок взят из письма Н. Ушакова (г.Ангарск) - см. рис.1.

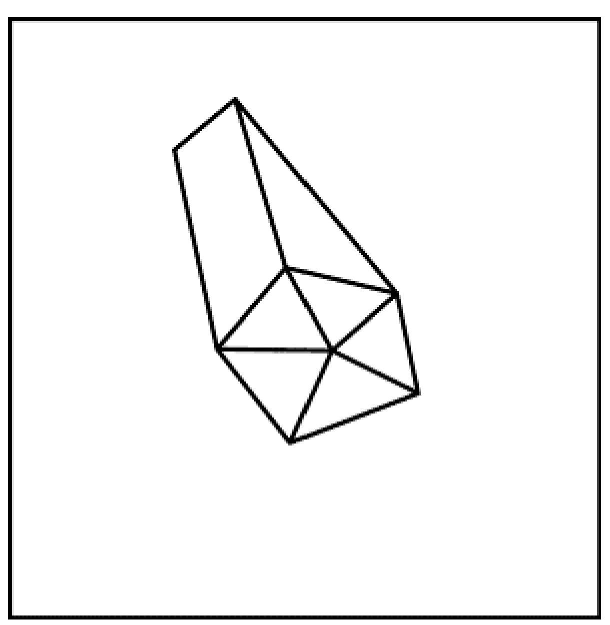


Рис.1

4. Есть разночтения по поводу подзарядки топливом от звезды. Мы не зря поднимали эту кажущуюся тривиальной проблему, 90% делают это не задумываясь, а 10% не могут сделать этого вообще. У них перегревается корабль задолго до включения надписи "Fuel scoops on" или просто температура не растет, но и заряда нет.

Может быть стоит прислушаться к мнению Михайлова С.Ю. (г. Саратов), который пишет, что "Cobra MK III" имеет нерегулярную геометрическую форму и поэтому есть разница, каким боком Вы ориентированы к звезде. Корабль, повернутый плоскостью, нагревается гораздо сильнее, чем корабль, развернутый торцом. "

Чтобы окончательно решить этот вопрос, пусть нам напишут только те, у кого заправка

никак не идет и сообщат о том, какой версией игры они пользуются. Может быть выяснится, что она у всех одна и та же.

В последнем выпуске (11-12) прошлого года мы сообщили о том, что пилоты начали исследования 102-байтного блока состояния, который отгружается, когда Вы сохраняете программу на ленте. Сегодня Радзевич А. А. из г. Нальчика дает дополнения к ранее неустановленным байтам.

15 - Ваш рейтинг.

16...18 - выполненные миссии.

41 - грузоподъемность (тонн).

61...66 - расположение звезд в галактике.

67 - ?

68 - координата у корабля.

69 - ?

70 - координата х корабля.

Интересное исследование провел пилот класса ELITE из г. Екатеринбурга Д. Палтусов.

Во-первых, он принудил свою версию программы работать с джойстиком, хоть она и очень не хотела:

```
POKE 29646,191: POKE 29649,226: POKE 29056,0: POKE 29659,0
```

(у него версия, помеченная Джеком О' Лантерном).

Во-вторых, он изучил машинный код программы и ему удалось установить кое-что интересное. Как оказалось, программа ведет внутри себя собственный подсчет очков.

За каждые 256 очков выдается сообщение RIGHT ON COMMANDER.

А вот как оценивается Ваша боевая активность при уничтожении кораблей:

THARGON - 4 балла.

FER-DE-LANCE - 3 балла.

ASP MK II - 3 Балла

PYTHON, ADDER-2, COBRA MK III, SIDEWINDER и "отшельник" - по 1 баллу.

Контейнер, астероид и VIPER - не дают очков.

Установлено соответствие и между набранной Вами суммой баллов и Вашим боевым рейтингом:

РЕЙТИНГ	ОЧКИ
HARMLESS	0
MOSTLY HARMLESS	9
POOR	17
AVERAGE	33
ABOVE AVERAGE	65
COMPETENT	129
DANGEROUS	768
DEADLY	2816
ELITE	6656

Кстати, это ответ тем читателям, которые недоумевают почему у них останавливается наращивание рейтинга после достижения COMPETENT.

Исследовав карты галактик, наш читатель обнаружил и невидимые звезды. Как и предполагалось ранее, это двойные звезды, получаемые наложением по OVER 1. Автору удалось просчитать их количество и расположение.

ГАЛАКТИКА	ЗВЕЗДЫ	
2	DIMAATMA	- ZAXERICE
4	QUZAARAR	- RIUSBEQU
5	EDXEBERE	- GEDIESQU
	LAZAZO	- ZAENZA
	TEZABI	- DIVEES
	ORLAED	- TETIRI
7	DICELASO	- ZAANXEGE

	ERVEAN	- SOINSOAN
8	CECEES	- ESUSALE

Есть еще одна пара в 8-ой галактике, но найти ее пока нашему корреспонденту не удалось.

Итоги конкурса на лучший технологически развитый маршрут.

Читатели 1991 года помнят, что в N 10 "ZX-РЕВЮ" мы предложили пилотам игры "ELITE" составить для любой галактики план наиболее технологически развитого маршрута из 20 пунктов. Сумма технологических уровней в нем должна быть максимальной, сегодня мы можем подвести итоги и объявить пять победителей.

Всего на конкурс поступило около 30 маршрутов, как правило, пилоты, приславшие их, имеют рейтинг "DEADLY" ИЛИ "ELITE". Да это и не удивительно, маловероятно, чтобы начинающий пилот смог облететь восемь галактик и найти наиболее интересные с этой точки зрения планеты.

Как оказалось, удача в поисках сопутствовала тем, кто удачно определился в выборе галактики для прокладки своего маршрута. Наиболее широко были представлены галактики N1, 2, 3, 7, 8 и вот каковы средние результаты в этих галактиках:

N1 - 198
N2 - 219
N3 - 206
N7 - 240
N8 - 219

Нет ничего удивительного, что те пилоты, которые выбрали для исследования галактику №7 оказались в наиболее выигрышном положении.

Вот первые 5 маршрутов-победителей (все они проложены в седьмой галактике).

1.

Пилот: Марченко А.
Порт приписки: г. Новая Каховка (Херсонской области).
Боевой рейтинг: "ELITE"
Боевой стаж: 2.5 мес.
Маршрут:

BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - ZARAUSXE -AGEBI - ESZAUSVE - ANGERIRI - QUGEZA - ESDITI - DIUSACE - ESRIXEAR - TIREGEES - ORESATRA - QUDIOR - ATESLETE.

Суммарный технологический уровень маршрута 251 балл.

2.

Пилот: Минеев А. В.
Порт приписки: Владивосток
Боевой рейтинг: not supplied
Боевой стаж: not supplied
Маршрут:

BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - ZARAUSXE - XEVEXEAN - AGEBI - ESZAUSVE - QULAAON - ANGERIRI - QUGEZA - ESDITI - AEDEDLE - DIUSACE -TIREGEES - ESRIXEAR

Суммарный технологический уровень маршрута - 248 баллов.

3.

Пилот: Ветлянский А. В.
Порт приписки: Анадырь
Боевой рейтинг: not supplied
Боевой стаж: not supplied
Маршрут:

BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - USQUEM - ZARAUSXE - AGEBI - ESZAUSVE - ANGERIRI - QULAAON - ISARE - ESDITI - DIUSACE - AEDEDLE - ESRIXEAR - TIREGEES

Суммарный технологический уровень маршрута - 248 баллов.

4.

Пилот: Сытник В. А.

Порт приписки: г. Угледар, Донецкой обл.

Боевой рейтинг: not supplied

Боевой стаж: 3.5 года

Маршрут:

BIONBIIN - MARAUS - CERIASON - ENARINES - ONANORRA - XEUSREOR - MARARERE - USQUEN - ZARAUSXE - AGEBI - ISARE - QULAAON -ESZAUSVE - ANGERIRI - QUGEZA - ESDITI - DIUSACE - AEDEDLE - ESRIXEAR - TJREGEES

Суммарный технологический уровень маршрута - 246 баллов.

5.

Пилот: Старцев А. Г.

Порт приписки: г. Миасс, Челябинской обл.

Маршрут:

TEABI - ERENRA - XEGEARER - BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - USQUEN - ZARAUSXE - ISENES - ISARE - QUGEZA - QULAAON - AGEBI - ANGERIRI - ESZAUSVE

Суммарный технологический уровень маршрута - 242 балла.

Как Вы можете убедиться, все маршруты-победители имеют немало общих пунктов. Таким образом, можно считать, что наиболее технологически развитая область в программе ELITE установлена с достаточной точностью.

В соответствии с условиями нашего конкурса все пилоты прислали вместе с маршрутами выписки из бортжурнала, в которых указано какие товары целесообразно покупать на каждой из планет, и вот, что мы установили:

Посетив 100 планет, участники конкурса 29 раз проголосовали за приобретение компьютеров и 19 раз - за меха. Прочие товары уступают по своей эффективности. Вывод очень прост: хотите иметь меха - осваивайте компьютеры. Эта мысль может иметь далеко идущие последствия.

Победителям мы высылаем ксерокопию повести "THE DARK WHEEL", написанной по мотивам программы "ELITE".

Теперь новое предложение:

Мы предлагаем проложить наиболее рискованный маршрут. Условия те же. В любой галактике облететь 20 планет, не побывав на одной планете дважды и подсчитать уровень опасности этого маршрута, давайте условимся, что за анархическую планету начисляется 5 баллов, за феодальную - 4 балла, а за любую другую - 1 балл.

Точно так же, как и в прошлый раз, просим прислать выписку из бортжурнала, в которой указано, какие товары наиболее целесообразно покупать в каждой из точек вашего маршрута, чтобы иметь в очередном пункте максимальную прибыль. Эти данные будут свидетельством того, что Вы действительно пролетели по маршруту, а не взяли его из Всегалактического справочника.

Кроме того, интересно узнать, а на чем же делают бизнес в самых злачных уголках вселенной.

Задача несравненно более трудная, по сравнение с той, что была поставлена в прошлый раз. Мы знаем, что у этих планет Вас будут подстерегать пиратские флотилии, но зато и ценность ваших наблюдений будет максимальной. Начинающим пилотам можно будет выдать рекомендации о том, в какие районы им не следует показываться, а опытный

бойцам высокого класса будет ясно, где наиболее быстро можно поднять свой рейтинг.

Для документальности Ваших отчетов Вы можете сообщить свой боевой рейтинг и стаж полетов. Будет приятно также напечатать о количестве одержанных Вами побед во время прохождения маршрута.

Призы те же - 5 ксерокопий повести "DARK WHEEL" и публикация отчетов в майском/июньском выпуске.

Отчеты принимается до 1.05.92 по почтовому штемпелю даты отправки.

POKES

Д. Палтусов, который только что поделился своими результатами исследования ELITE активно исследует и другие программы. Вот список некоторых POKES, которые ему удалось отыскать самостоятельно:

TUTANKHAMUN	27783,0
MOTOS	48241,0
LODERUNNER	35427,24: 35426,0
DOWN TO EARTH	40142,195
SANXION	36584,0
KNIGHT LORE	53567,0: 50205,0: 50206,0: 50207,0:
SPELLBOUND	27038,8 (в начале)
BATTY	48437,167
CHRONOS	53407,N
BOULDER 4	30960,N
COMMANDO	27652-27657,0
BOULDER 1	31007,0: 31008,0: 31009,0
METAL ARMY	48535,0: 48559,0
	(энергия)
	42198,0: 48700, 107
	(жизни)
KRAKOUT	46565,0: 61534,0
N.O.M.A.D.	40167,0
RENEGADE	41047,36
ACADEMY	31378,N: 31386,N
	31249,N: 31305,N (N-оснастка корабля)
EQUINOX	41914,0
HYSTERIA	44588,201
L. NINJA 2	36579,175:36578,0
GUNFRIGHT	49233,54: 49234,1: 49235,0: 49236,0
ASTEMEX	43516,0
AIR FORCE II	51904,0
CIBERNOID	39402,0
CIBERNOIDD 2	36197,0
FRED	31171,0
IMPACT	54500,183

Два слова о том, как ищутся адреса для POKES на примере программы N.O.M.A.D

Зная, что в программе в исходном состоянии игрок имеет 3 попытки, наш читатель предположил, что где-то есть ячейка памяти, в которой хранится это число. Можно также предположить, что засылается оно туда в начале работы программы через регистр A. Это, конечно не единственный способ засылки числа в память, но он достаточно удобен и широко распространен.

Просмотрев с помощью дисассемблера программу, он сразу отбросил те области, в которых хранится графика, спрайты, тексты и т.п., сосредоточившись на машинном коде. (Как можно всего за несколько минут прикинуть где что в программе находится, "ИНФОРКОМ" писал во 2-ом томе трехтомника по программированию в машинном коде.)

Такой беглый просмотр позволяет выделить для более подробного исследования область длиной всего в несколько Кб. В этой области ищется команда Ассемблера LD A,3. Это занимает еще несколько минут. Если таких точек несколько, их надо будет все испытать, что делается следующим образом.

Рассмотрим фрагмент программы:

```
..............
LD A,3
LD (34586),A
LD (34480),A
LD HL,27304
LD (34584),HL
..............
```

Итак, есть подозрение, что в ячейке 34536 или 34480 организована переменная, в которой хранится количество попыток.

Давайте проверим ячейку 34586.

С помощью поисковых возможностей ДИСАССЕМБЛЕРа найдем где еще упоминается этот адрес и посмотрим, что там записано. Обычно это выглядит так:

```
.............
LD A,(34586)
DEC A
LD (34586),A
.............
```

Команда DEC A, уменьшающая значение в аккумуляторе на единицу, - прекрасный кандидат на то, что мы ищем. Ведь после гибели героя количество попыток уменьшается на одну.

Проверим это. Заменим DEC A на команду NOP (нет операции - ее код =0), и запустим программу после этой переделки. Если мы попали правильно, Ваш герой теперь бессмертен.

А вот как ищутся пароли для игры. Возьмем программу SABOTAGE, пройдя первый уровень удалось установить пароль второго (его программа выдала сама) - "BUMBLE BEE 2".

Теперь найдем место в программе, где хранится эта фраза, можно даже не пользоваться ДИСАССЕМБЛЕРом, а сделать это из БЕЙСИКа.

Код буквы "B" равен 66, а код буквы "U" - равен 85.

```
10 FOR i=25000 TO 65535:
      IF PEEK i = 66 AND PEEK (i+1) = 85
      THEN PRINT i
20 NEXT i
```

Когда будет найдено место в программе, в котором имеется сообщение "BU... ", компьютер выдаст вам его адрес, проверьте близлежащие адреса. В них может быть тоже что-либо интересное, так удалось установить пароли прочих уровней:

2) BUMBLE BEE 2.
3) HONORARIUM.
4) PHENOMENON.
5) ONOMASTICS.
6) SALMAGUNDI.
7) PSEUDONIMOUS.
8) ONOMATOPEDIA.

Мы благодарим Д. Палтусова за интересную информацию и вынуждены только пожалеть, что в ELITE COMPETITION он не занял призового места. Он нашел великолепный технологически развитый маршрут для галактики N1, но не догадался исследовать галактику N7, как это сделали победители.

Кстати, о паролях. Наш читатель Жукович Н. Н. из г. Усолье-Сибирское очень просит помочь ему найти пароль 8-го уровня в игре IMPACT. Может быть кому-то удалось его пройти? Для тех же, кто хочет попробовать свои силы в этой увлекательной игре, он сообщает пароли прочих уровней.

В игре 9 уровней по 10 экранов:

Экран 1 - пароль не требуется,

Экран 11 - EGGS
Экран 21 - CHIP
Экран 31 - LEAD
Экран 41 - TICK
Экран 51 - CASE
Экран 61 - FACE
Экран 71 ????
Экран 81 - USER - пользовательский.

Наш читатель из г. Смоленска, Коновалов А. В. тоже активно работает с POKES. К сожалению, он отмечает, что некоторые POKES из тех, что были опубликованы в №10 у него не пошли. В то же время, известно, что по стране ходят многочисленные версии даже для одной и той же игры. Поэтому для тех, у кого не пошли какие-то POKES, он предлагает воспользоваться прилагаемым ниже набором.

Все прилагаемые POKES вставляются после последнего LOAD"", а не между USR.

GAMEOVER1	39334,0 (жизни)
	32417,0 (гранаты)
HORACE AND SPIDERS	27680,60
MAG-MAX	58472,60
MUTANT MONTY	55761,60: 56483,183
CYBERNOID-1	39403,0
VIXEN-III	41423,0
FIREBIRDS	27235,0
XEVIOUS	53592,n
BLADE WARRIOR	37161,0
	39490, 60
KARNOV	25620,0
GAMEOVER 2	38704, 0 (жизни)
	32388,0 (гранаты)
GONZZALEZZ 2	35749,0 (жизни)
	48056,0 (оружие)
ASTEMEX	43517,52
GONZZALEZZ 1	37085,0
METAL ARMY	36697,0 (жизни)
	42650,0 (энергия)
ACTION FORCE 2	47874,182
KRION	45004,0 (жизни)
	44486,0 (горючее)
GLUG GLUG	34139,0
PSSST	24984,0
ROGUE TROOPER	30924,0
EQUINOX (для версии ROBY'86)	39858,52:39659,182 (жизни)
	48691,0:48762,0 (горючее и лазер)
STAINLESS STEEL	46957,60
UNDER WURLDE	59376,0
RICK DANGEROUS	55460,0 (жизнь)
	61045,0 (гранаты)
	60954,0 (патроны)
CAVELON	47250,0
R-TYPE	30873,0
DAN DARE 2/1	46459,0
VIGILANTE	48735,60
KOKOTONI WILF	43742,0
PETER PACK-RAT	27243,100

PENTAGRAM	49917,0
THUNDERBLADE	33199,0
	33145,0
THOR	37550,195
DAN DARE 2/2	51797,0
MOTOS	42241,0
CHICAGO	60264,0
FROST BYTE	30992,183 (жизнь)
	28237,183 (время)
RUFF & REDDY	33567,0
PROJECT FUTURE	29332,0
JETMAN	36965,0

Комментарий "ИНФОРКОМа".

Дорогие друзья! То, что игры имеющие хождение в стране, уже давно не являются первозданными, Вам хорошо известно. Сначала их курочат в Голландии, потом через ФРГ это поступает в Польшу, где дело поставлено не хуже, чем у нас, и только потом мутными ручейками программы поступают на расправу к нашим специалистам.

Вам это известно очень хорошо.

Кроме того, для различных POKES существует и разная технология их введения, что тоже немаловажно и не всегда правильно выполняется. Радикальных рецептов быть не может, кроме одного: учится, учится и учится, как завещал сэр Клайв Синклер, а в Англии, как Вы понимаете, звание сэра и титул лорда простому инженеру за одни красивые глаза не дают.

Осваивайте машинный код, изучайте опыт и приемы тех, кто это уже освоил, развивайте собственные приемы. В принципе, на страницах "ZX-РЕВЮ" мы даем достаточно информации, чтобы было с чего начать. Как Вы увидите, в этом году в "ZX-РЕВЮ-92", будет немного больше материалов для тех, кто осваивает машинный код.

Только освоив технологию Вы почувствуете себя хозяином положения и сможете воспользоваться чьим-то POKE с полным знанием дела или адаптировать его под свою конкретную программу.

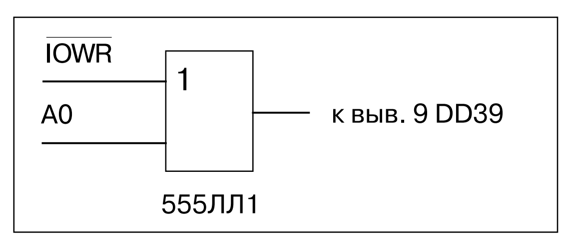
Вопросы совместимости.

Мы продолжаем рассматривать вопросы совместимости отечественных моделей ПК с фирменным программным обеспечением. Наши читатели уже встречались с разработками в этом направлении, ведущимися Полубарьевым Сергеем Викторовичем из г. Йошкар-Ола.

Сегодня речь идет о наведении порядка с дешифрацией портов ввода/вывода в версии Зонова (15 корпусный "Ленинград-1") для обеспечения полной совместимости с оригиналом.

Вот последовательность доработки. Для этого необходимо установить на свободном месте платы по одной микросхеме К555ЛЛ1 и К555ЛИ1 и далее выполнить в схеме следующие изменения:

1) Отключить сигнал IOWR от вывода 9 DD39 (К555ТМ9 все обозначения по принципиальной схеме "Ленинград-1") и собрать на одном из элементов К555ЛЛ1 следующую схему:



После этого запись в порт вывода будет происходить только по четным адресам, что исключает мерцание и самопроизвольное изменение цвета бордюра.

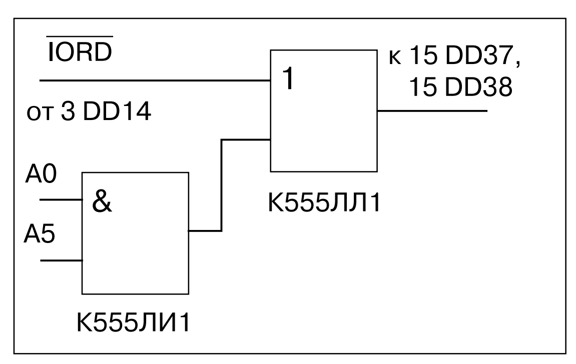
Небольшое отступление перед последующими доработками:

"Ленинград-1", по-видимому, разрабатывался под самодельный джойстик с нормально-замкнутыми контактами. Наиболее же распространенными являются джойстики "Atari"-типа, у которых контакты нормально разомкнуты и замыкаются при отклонении рукоятки. Часть последующих переделок будет направлена на обеспечение совместимости именно с ними.

2) Резисторы R21...R25 сопротивлением 15 кОм, подключенные к линиям джойстика DV0...DV4 соответственно, заменить на МЛТ-0,125 с сопротивлением порядка 660 Ом...1.5 кОм. Общую точку их соединения отключить от шины "+5V" и соединить ее с "землей".

3) Свободные входы DD37 (выводы 6,10,13) соединить с выводом 8 этой же микросхемы.

После выполнения этих доработок в принципе уже можно включить компьютер и оценить результаты. Команда PRINT IN 31 должна при отпущенном джойстике печатать 0, а при отклонении рукоятки или нажатии кнопки - некоторое число, зависящее от положения джойстика. Запись любого числа по любому нечетному адресу OUT не должна изменить цвет бордюра. Однако, если Вы хотите довести джойстик до полного Кемпстон-стандарта, а заодно и освободить часть нечетных портов ввода, необходимо сделать еще одну доработку:



4) Отсоедините сигнал IORD, поступающий с вывода 3 DD14 (K555ЛЛ1) от вывода 15 DD37 и DD38, и соберите следующую схему его дополнительной доработки:

Теперь считывание из портов будет производится так:

A0=0 и A5=1 - клавиатура;

A0=0 и A5=0 - клавиатура; т.е. клавиатура читается по любому четному адресу порта.

A0=1 и A5=0 - Кемпстон-дж-к.

A0=1 и A5=1 - несуществующий порт, т. е. все нечетные порты с адресами, большими, чем 31 свободны.

Практика показывает, что при выполнении доработок 1...3 большинство проблем исчезает. Доработка 4 не является обязательной (в игровых программах обычно эффекта не дает) и применяется во-первых, чтобы обеспечить соответствие стандарту и, во-вторых, чтобы освободить часть адресного пространства, например для установки "ZX-LPRINT III".

Соответственно, если необходимости в ней нет, микросхему K555ЛИ1 можно не устанавливать, оставшиеся свободными элементы дополнительных ИМС можно использовать, например, при подстыковке интерфейса "BETA-DISK".

Если рассмотреть доработку, предлагавшуюся в ZX-РЕВЮ в прошлом году (стр. 157, рис. 7), то можно понять, что в принципе она делает то же самое, но при этом со старших 3 битов порта 31 продолжает считываться единица, что в принципе может в некоторых случаях приводить к нарушению совместимости и в некоторых играх и приводит.

Теперь несколько слов об отмечавшемся замедлении работы игровых программ на "Ленинграде-1" (см. ZX-РЕВЮ-91. с. 197).

Прежде всего, примем за аксиому следующее:

а) Схема построена так, чтобы использовать кварцы от 13.0 до 14.5 МГц. Тактовая частота процессора получается при делении частоты генератора на 4, т.е. это 3.5 МГц при частоте кварца 14 МГц. Поэтому первым делом проверьте кварц, который стоит в Вашей машине.

б) Конфликт дисплея и процессора (о котором писали на странице 52 "ZX-РЕВЮ-91") может происходить по любому адресу ОЗУ, поскольку у "Ленинграда-1" сплошное поле памяти 48K. В этом случае процессор приостанавливается по сигналу WAIT, формируемому на триггере D 9.2. Исходными для формирования служат сигналы M1 процессора и CSRAM (выборка ОЗУ поступает на вывод 1 регистра K555ИР22). Однако, в некоторых схемах и рекомендациях по наладке (соответственно и в платах) вместо сигнала CSRAM требуют использовать сигнал MREQ. Последствия этого - замедление работы даже с ПЗУ (хотя на некоторых экземплярах компьютера это, возможно, и необходимо).

Методы устранения несложны.

а) Замените кварц на 14,0 или 14,5 Мгц. Чтобы при этом не нарушалась синхронизация TV, внесите изменения в схему включения счетчика D4 (K555ИЕ7) согласно следующей таблицы:

Частота, МГц	Выводы D4 соединить с:	
	"+5 вольт"	"общий"
13,0	10, 15	1,9,14
13,5	10	1,9,14,15
14,0	1, 15	10,9,14
4,5	1	10,9,14,15

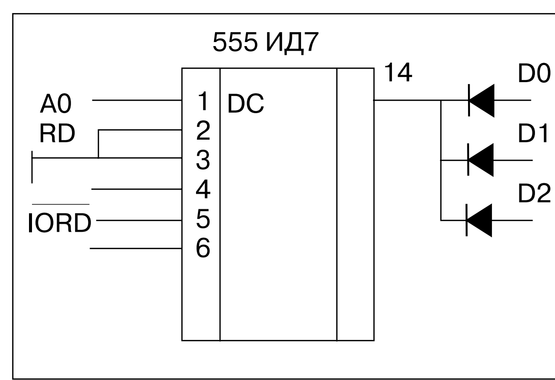
Если после замены кварца обнаружатся сбои ОЗУ, придется поменять микросхемы на более быстродействующие или попытаться подстроить временные диаграммы сигналов RAS и CAS.

6) Решение второй проблемы вытекает из самой проблемы, однако, если после замены на формирователе MREQ на CSRAM обнаружатся сбои ОЗУ, придется вернуть все на место.

В любом случае рекомендуется после проведения доработок выполнить длительную серию тестов ОЗУ, чтобы убедиться в надежной работе компьютера.

Все предлагаемые доработки проверены автором лично на практике, во избежание появления сбоев лучше применять в компьютере "Ленинград-1" только серию K555 (K1533), не заменяя ее на K531 и K155 - тогда наладка почти не требуется.

Для тех, кто работает с Краснодарским и Харьковским вариантами компьютера, впрочем как и с "Ленинград-1", наш читатель из Харькова В.И. Бойко предлагает несложную доработку, обеспечивающую работу таких программ, как "ARKANOID", "DUET" и некоторых других.



На ножку 6 микросхемы подается старший байт адреса, формируемый схемой генерации адреса ULA. Все остальные сигналы можно взять непосредственно с процессора.

Просьба о помощи.

В Межшкольном комбинате г. Майли-Сай (Кыргызстан) установлена локальная сеть на базе "Спектрумов", изготовленных в Ташкенте. Проблема с эксплуатацией программ в сети. Программы есть, но по сети не идут. Если у кого есть опыт эксплуатации сети, просьба отозваться. "ИНФОРКОМ" рассматривает вопросы обеспечения работы школ, как самые приоритетные.

Контактный адрес: 715420, Кыргызстан, Джалал-Абадская область, г. Майли-Сай, ул.

Маяковского. 2, кв. 21, Курику В. Н.

Сам "ИНФОРКОМ" располагает всей необходимой информацией по тому, как организуется локальная сеть для фирменных машин на базе интерфейса локальной сети ZX-INTERFACE I. Известна система команд, системные переменные, коды-перехвата, но мы совершенно не знаем, на каких интерфейсах делают сети у нас и какими командами они задействуются.

Примем к печати статьи на данную тему.

И снова SHERLOCK.

Нашим читателям уже известны проработки эксперта Ескевича А. А. из г. Новосибирска. Как студента-филолога, его конечно интересуют текстовые игры (жанр adventure) и он активно взялся за работу с программой SHERLOCK, но столкнулся с проблемой.

Узнав от Уотсона о том, что в Лизерхэде (LEATHERHEAD) произошло убийство двух человек, он решил направить Холмса на место преступления. Проблема в том, что ни один извозчик не желает его понимать. Он побывал на всех четырех вокзалах и на прилегающих к ним улицам (Aldereate street, Bishops Road, Backingham Palace и King Cross), но ни один извозчик ниоткуда не желает везти его в Лизерхэд.

Мы с большим удовольствием принимаем эту жалобу на безобразную работу лондонских извозчиков и должны заметить, что у нас в Москве тоже очень часто бывает, что таксисты не хотят Вас везти ниоткуда и никуда. Не знаем, как обстоят дела в Новосибирске. С другой стороны, есть сведения о том, что некоторым удается как-то договорится с водителем, возможно, есть какие-то филологические тонкости в общении пассажира и водителя нам неведомые.

Попробуйте применить такой же подход и к Лондонским кэбби. Одним словом: "Вам нужно найти необходимые слова". Вот они:

Выйдя из дома, Холмс спешит на утренний поезд в Лизерхэд, который отправляется в 9:15. Для этого ему нужно нанять кэб:

1. HAIL CAB
2. CLIMB INTO CAB
3. SAY TO CABBY "GO TO..."

Вот и все. Этим же методом Вы будете нанимать кэб и в других подобных ситуациях.

На платформе Холмс встретит инспектора Лестрейда... а далее желаем Вам успеха.

А вот, что пишет Антон Скворцов из С. Петербурга. "Меня сразу заинтересовали ваши статьи об адвентюрных играх. Мне они во многом помогли. Правильно, что Вы развиваете интерес к этому жанру.

Я. правда, еще не очень опытен и хотел бы переписываться с кем-нибудь, кто любит эти игры. Проблема в том, что на нашем "толчке" их не достать, и я пока довольствуюсь двумя программами - "Knight-Tyme и The Hobbit. "

Уважаемый Антон, развивая это направление, мы сознательно шли против течения, против моды, против законов местных "толчков". Да, сегодня энтузиастов этого жанра пока мало. Но есть мировой опыт, и он гласит однозначно - через два-три года общения с компьютером основными становятся игры трех жанров - ADVENTURE, STRATEGY, MANAGMENT. Так что не беспокойтесь, если на Вашем "толчке" есть умные предприниматели, они это поймут, перестроятся и в течение года - двух ситуация резко изменится.

Наша задача - подготовить этот поворот, а Вы правильно делаете, что осваиваете это направление. Когда придет его время, будете уже иметь нужный боевой (а может быть и деловой) опыт.

Для любителей "умных" игр, желающих связаться с Антоном: 195027, С.-Петербург, Среднеохтинский проспект, д. 2 "А", кв. 52.

Внимание! Будьте осторожны.

Наш читатель из г. Харькова Хоминич Р. В. сообщает о неприятном моменте, который поджидает многих любителей аркадных адвентюр, начавших игру в программу BLACK RIDER фирмы TOPOSOFT,1988.

Прежде чем начать с ней работать, желательно проверить ее на полноту состава файлов.

А вот в чем суть.

Вы являетесь капитаном пиратского корабля, корабль находится на стоянке в бухте и ремонтируется. С каждым днем становится все труднее управлять разлагающейся командой, матросы пьют ром и готовы разбежаться.

Ваша первая задача - сохранить команду. Разрушить сходной трап можно выстрелом из пушки, но надо найти фитиль (факел).

Поиск ведется в многочисленных сундуках с сокровищами, расположенных в различных помещениях корабля. Чтобы не тратить время на возню с замками, капитан открывает их не церемонясь - выстрелом из пистолета.

Среди прочего барахла в этих сундуках Вы найдете и полезные для себя веши.

Все время Вам придется отбиваться от своих подгулявших подчиненных. Интересна система подкрепления энергии. Подкрепиться можно ромом, но будьте осторожны, не переберите.

Когда Вы обыщете все сундуки, на самой нижней палубе Вас будет ждать сюрприз, старый красный сундук, который до этого никак не открывался, вдруг окажется открытым, а рядом с ним лежит уникальная карта острова сокровищ. Вы бросаетесь к ней и… программа просит загрузить следующий уровень, что очень неприятно, если у Вас его нет, а именно в таком виде программа и нашла широкое распространение.

Тех счастливых джентльменов удачи, которые владеют полной игрой, мы просим прислать список ее файлов с указанием длины и названия, дабы многочисленные поклонники аркадных адвентюр не становились жертвами столь бездушного обращения.

"РЕГИСТРАТУРА" - система многоцелевого назначения.

Система относится к разряду информационно-поисковых и справочных систем, исполняемых и настраиваемых "под заказчика". Данная система может удовлетворить большинство потребностей организаций в удобном программном средстве для хранения и обработки текущей информации.

I. НАЗНАЧЕНИЕ СИСТЕМЫ

Система предназначена для ведения учета и проведения обработки любой информации, необходимой на Вашем предприятии (в Вашем учреждении).

Например:

- регистрация входящих и исходящих документов;
- регистрация входящих и исходящих товаров;
- регистрация пациентов в лечебном учреждении;
- регистрация клиентуры и заказчиков
- и многое, многое другое.

Система может удовлетворить потребности отдела кадров, планово-финансового отдела, отдела соцкультбыта, различных административных, муниципальных и хозяйственных служб и т.п., КРОМЕ БУХГАЛТЕРИИ.

II. ВОЗМОЖНОСТИ СИСТЕМЫ

Система позволяет:

- вводить информацию в любое заданное количество полей;
- просматривать информацию в избранном формате;
- вносить изменения;
- сортировать информацию по любому полю или по любой их совокупности;
- производить поиск по заданному критерию или по любой их совокупности;
- произвольно по каждому критерию задавать ключевое соотношение на поиск;
- получать статистическую информацию (производить подсчет записей, удовлетворяющих любым заданным критериям при любых заданных соотношениях);
- распечатывать всю или избранную информацию на принтере в избранном составе, избранным шрифтом на листах избранного формата;
- делить избранные файлы на части в автоматическом или ручном режиме;
- объединять файлы;
- .... и многое другое.

Практически все операции, включая выбор формата просмотра, выбор критериев на поиск и сортировку, выбор ключевых соотношений и т. д. автоматизированы и не вызывают трудностей у неподготовленного пользователя. Система имеет приятный дизайн, удобную, не вызывающую утомления систему управления 6-ю клавишами, доставляет радость в работе.

Основной принцип: пользователь должен только ввести информацию в первый раз - все остальное выполняется простейшим выбором из меню альтернативных предложений.

Несмотря на то, что освоение системы неподготовленным пользователем может проходить на интуитивном уровне, к ней прилагается краткая инструкция, гарантирующая освоение системы в считанные часы.

Эта система - наилучший ответ на Ваши самые срочные потребности. Хотите немедленно ощутить мощный эффект от внедрения ЭВМ в Вашу организацию - воспользуйтесь этой системой.

Введенная Вами информация не пропадет и при переходе в дальнейшем на другие, более мощные системы, в том числе и работающие в локальных сетях. Вся введенная Вами информация хранится в формате .DBF, который во всем мире признан неофициальным стандартом.

III. КОМПЛЕКТ ПОСТАВКИ

Система поставляется на одной дискете 5.25" (MS DOS, 360 K). На титульном экране системы проставляется название Вашей организации.

IV. ИСХОДНАЯ ИНФОРМАЦИЯ

Поскольку система исполняется персонально "под заказчика", от Вас необходимо получить исходную информацию - заверенный руководителем СПИСОК НЕОБХОДИМЫХ ВАМ ПОЛЕЙ ИНФОРМАЦИИ И ИХ РАЗМЕР, например:

1. ФАМИЛИЯ	- 16 знаков
2. ИМЯ	- 12 знаков
3. ОТЧЕСТВО	- 16 знаков
4. ДАТА РОЖДЕНИЯ	- 8 знаков
5. ДОМ. АДРЕС	- 40 знаков
6. ПОЛ	- 1 знак
....	
5. ПРИМЕЧАНИЕ	- 20 знаков

Просим Вас:

1. Не задавать размер полей более 40 знаков. Например, поле адрес можно разделить на 3 поля ПОЧТ. ИНДЕКС, АДРЕС, ТЕЛЕФОН.
2. Не задавать названия полей длиннее 16 знаков. Пользуйтесь сокращениями. Например: ДАТА ОКОНЧ. ВУЗА.

V. СРОК ИСПОЛНЕНИЯ ЗАКАЗА.

Срок исполнения - 2 - 3 недели после поступления средств на наш р/с и заказа с заверенным СПИСКОМ ПОЛЕЙ ИНФОРМАЦИИ.

VI. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К АППАРАТНО-ПРОГРАММНОМУ ОКРУЖЕНИЮ

1. Полная аппаратно-программная совместимость с IBM PC XT/AT. Надежность функционирования на отечественных модификациях не гарантируется и не обсуждается.
2. Наличие "жесткого" диска ("Винчестера") стандартного объема.
3. Дисковод гибких дисков 5,25" или 3.5"". Эта система поставляется нами без защиты от копирования.
4. Операционная система - MS DOS не ниже 3.20.
5. Русификация компьютера в стандарте ГОСТ (кодировка альтернативная).
6. Требования к монитору - не специфицируются, желательно - EGA.
7. Требования к принтеру - совместимость со стандартом EPSON.

VII. ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА

Гарантийным свидетельством при поставке программного продукта является картонный альбом, в который вложены дискеты с указанной на нем датой продажи. При его отсутствии поставка выполняется с заверенным гарантийным талоном.

Гарантиями обеспечивается:

- бесплатная замена поставочных дисков, неработоспособных в состоянии поставки (в течение месяц после поставки);
- замена с минимальной оплатой при выходе программ из строя по вине пользователя (механическое или электромагнитное повреждение, поражение вирусом на машине пользователя и т. п.) или по истечении месяца после поставки. Минимальная оплата не превышает стоимости дисков + 5% текущей стоимости программного обеспечения + стоимость почтово-транспортных расходов и согласовывается с потребителем.

VIII. ПОРЯДОК ОФОРМЛЕНИЯ ЗАКАЗА.

а) Направить в наш адрес письмо заказ с указание необходимого программного продукта и количества копий. Приложить копию платежного поручения и заверенный СПИСОК ПОЛЕЙ ИНФОРМАЦИИ И ИХ РАЗМЕР.

Наш адрес: 107241, Москва, Б-241, а/я 37, "ИНФОРКОМ"

б) Произвести предварительную оплату платежный поручением на наш р/с: N 500461778 во Фрунзенском коммерческом банке г. Москвы.

Стоимость системы на период март-апрель 1992г. - 7200 рублей + 28%

"ЗЕЛЕНЫЙ ПАКЕТ" ДИСТРИБУТОРА

О том, что такое "зеленый пакет" Вы можете подробно прочесть в N11-12 "ZX-РЕВЮ" за 1991 г.

Стоимость "зеленого пакета" по системе "РЕГИСТРАТУРА" составляет для частных лиц:

Рабочая версия программы - 10% * 7200 = 720
Дискета - 100
Почтовые расходы- 5

Итого 825 рублей

Уточняем, что высокая стоимость дискеты вызвана тем, что нашим дистрибуторам поставляются дискеты особо высокого качества с защитным тефлоновым покрытием, имеющие особый представительский вид и высокую коммерческую стоимость.

Напоминаем, что затраты дистрибутора на "зеленый пакет" являются только залогом и возвращаются по требованию. Активно работающим дистрибуторам затраты возвращаются при выплате комиссионных и далее такие пакеты предоставляются бесплатно.



МКП "ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

Уважаемые читатели!

Мы обращаем Ваше внимание на изменение нашего почтового адреса. Все почтовые отправления просим направлять по адресу 121019, Москва, Г-19, а/я 16, Те, кто связан с нами не первый год, знают, что с этим адресом мы когда-то начинали и сейчас возвращаемся к нему опять.

Это временная мера. Мы подготовим новый постоянный почтовый адрес и в следующем выпуске Вас оповестим.

В связи с этим мы вынуждены прекратить прием от Вас предоплаты за наши разработки. Все, что Вы сочтете нужным заказать, будет Вам высылаться наложенным платежом без предоплаты. Полностью также прекращается прием новых подписчиков. Если к концу года останутся нераспределенные экземпляры, предложим их наложенным платежом.

Мы также приносим Вам извинения за то, что Вы получите этот номер со значительной задержкой, вызванной сложностью с привлечением доступных нам по затратам полиграфических мощностей. Мы надеемся, что Вы простите нам эту задержку. Вместе с тем, мы еще раз просим Вас не беспокоиться, все 12 выпусков в 1992 Вы получите без каких-либо доплат, переподписок и т.п. Мы не исключаем возможности задержек, но все взятые обязательства выполним невзирая на известные экономические трудности страны.

СПЕКТРУМ В ШКОЛЕ

К УРОКУ ИСТОРИИ.

Эту программу можно использовать на уроке истории для проверки знаний. Вы, конечно, сами сообразите, как её можно адаптировать для других учебных дисциплин. Правильный ответ выбирается путём нажатия на клавишу от 1 до 4.

Учитель может увеличить количество вопросов по своему желанию, добавляя строки DATA (500,501,502,....9999). Следует помнить, что в последней строке DATA должно стоять "eof". (End of File - конец файла).

Мы использовали для диалога с пользователем русский шрифт, полагая, что Ваш компьютер русифицирован. Если он не имеет русифицированного ПЗУ, русифицируйте его программным путем, о чем мы неоднократно писали, например в работе "Большие возможности Вашего "Спектрума" ".

Программа будет также полезна начинающим для самостоятельного разбора, она достаточно проста для этой цели. Укажем только на небольшую особенность в строках 410 и 420.

Здесь Вам предлагается повторить игру нажать клавишу "Y". Поскольку неизвестно заранее, что именно нажмет играющий "y" или "Y", в строке 420 проверяется код символа, закрепленного за нажатой клавишей. Код "Y" равен 89, а код "y" равен 121. И в том и в другом случае выполняется переход к строке another (строка 110), после чего тест повторяется.

Вы можете усложнить эту программу по своему вкусу, например рандомизировав (сделав случайным) порядок следования вопросов. Можете ввести свою систему очков за правильный ответ с первой попытки. За правильный ответ со второй попытки и штраф за

неправильный ответ и т. п.

```
10 REM УРОК ИСТОРИИ.
20 LET data = 480:
      LET finish = 360:
      LET nextquest= 130:
      LET another=110
30 REM оформление экрана
40 BORDER 2: PAPER 7: INK 9: BRIGHT 1: CLS
50 PRINT PAPER 1; FLASH 1; AT 9,9 ;"УРОК ИСТОРИИ"
60 PAUSE 500
70 CLS
80 PRINT AT 4,2; "Эта программа служит для проверки Ваших знаний по истории. После каждого
      вопроса Вам предлагаются четыре возможных варианта ответа, только один из которых
      является верным. Выберите правильный ответ нажатием клавиши 1...4."
90 PRINT FLASH 1; AT 19,5; "Нажми любую клавишу"
100 PAUSE 0
110 REM another
120 RESTORE data: DIM r(3)
130 REM nextquest
140 CLS
150 READ a$
160 IF a$="eof" THEN GO TO finish
170 READ b$,c$,d$,e$,f$
180 PRINT AT 6,2; a$ ' '
190 PRINT TAB 4; "1. "; b$
200 PRINT TAB 4; "2. "; c$
210 PRINT TAB 4; "3. "; d$
220 PRINT TAB 4; "4. "; e$
230 FOR n=1 TO 2
240 PRINT AT 2,2; "Попытка ";n
250 PAUSE 0
260 IF INKEY$=f$ THEN PRINT PAPER 1; AT 19,13; FLASH 1; "ВЕРНО":
      LET r(n)=r(n) + 1: PAUSE 150: GO TO nextquest:
      REM: Правильный ответ
270 NEXT n
280 REM Неверный ответ после двух попыток
290 LET r(3)=r(3)+1
300 IF f$="1" THEN PRINT AT 18,7; INK 2; FLASH 1; b$
310 IF f$="2" THEN PRINT AT 18,7; INK 2; FLASH 1; c$
320 IF f$="3" THEN PRINT AT 18,7; INK 2; FLASH 1; d$
320 IF f$="4" THEN PRINT AT 18,7; INK 2; FLASH 1; e$
340 PAUSE 300
350 GO TO nextquest
360 REM finish
370 CLS
380 PRINT AT 6, 2: "Правильных ответов с первой попытки "; r(1)
390 PRINT AT 9, 2; "Правильных ответов со второй попытки "; r(2)
400 PRINT AT 15,2; "Неправильных ответов "; r(3)
410 INPUT "Попробуем еще раз?",y$
420 IF CODE y$ =89 OR CODE y$=121 THEN GO TO another
430 STOP
440 REM  ДАННЫЕ
450 REM  Введите столько дополнительных вопросов, сколько хотите в строки 503 и далее. За
      каждым вопросом должны следовать четыре альтернативных ответа и цифра, показывающая
      какой же из них является правильным. Список вопросов и ответов должен заканчиваться
      строкой DATA, к которой стоит запись "eof", как показано в нашем примере.
500 DATA
      "В каком году в России было отменено крепостное право ?", "1825",
      "1855","1861","1917","3"
501 DATA
      "Кто из русских царей одержал победу в Полтавской битве?",
      "Иван Грозный", "Петр Первый","Павел 1", "Николай 1","2"
```

```
502 DATA
      "Кто из русских царей был кавалером Мальтийского Ордена",
      "Иван Грозный", "Петр Первый", "Павел 1", "Николай 1" "3"
9999 DATA "eof"
```

POKES

NETHER EARTH

Наберите приведенный листинг и запустите его (RUN) перед загрузкой программы. Вы получите неограниченные ресурсы для создания боевых роботов. Будет неплохо, если Вы перед запуском этого блока отгрузите его на ленту, чтобы избежать необходимости повторного набора, если досадная опечатка испортит Вам всю работу.

```
1 BORDER 0: PAPER 0: INK 7
5 CLEAR 65535
10 PRINT AT 10,3; "START 'NETHER EARTH' TAPE"
20 LOAD "" CODE 64730
25 POKE 64753,254
30 FOR f=65024 TO 65036
40 READ a: POKE f, a: NEXT f
50 DATA 62, 18, 50 ,111, 173
60 DATA 62, 33, 50 ,71, 174
70 DATA 195, 0, 166
80 RANDOMIZE USR 64730
```

* * *

HEAD OVER HEELS

Очень рекомендуем Вам сыграть в эту замечательную программу. Ниже приведенный листинг позволит вам стать непобедимым и нормально исследовать все ее лабиринты. "ИНФОРКОМ" с удовольствием напечатает грамотно и художественно написанный отчет о Вашем путешествии.

Программа снабжена счетчиком контрольной суммы и сможет Вас предупредить, если при наборе Вы где-либо ошибетесь, но копию все же лучше сделать.

```
1 CLEAR 64500
2 LET t = 0: LET W=1
5 FOR f=32000 TO 32170
10 READ a: POKE f,a
15 LET t=t+a*w: LET w = w+1
20 NEXT f
15 IF t<>1764297 THEN PRINT "DATA ERROR": STOP
30 PRINT AT 10,1; "START 'HEAD OVER HEELS' TAPE"
50 RANDOMIZE USR 32000
100 DATA 221,33,203,92,17,234
110 DATA 6,62,255,55,205,86,5
120 DATA 48,241,243,237,94,33
130 DATA 44,125,229,33,173,98
140 DATA 229,51,51,17,163,252
150 DATA 1,22,3,33,253,94,62
160 DATA 202,237,79,195,173,98
170 DATA 33,70,125,229,33,199
180 DATA 252,229,51,51,17,209
190 DATA 252,1,232,2,33,209,252
200 DATA 62,196,237,79,195,199
210 DATA 252,33,209,252,17,209
220 DATA 138,1,92,0,237,176,33
230 DATA 228,138,34,233,138,34
240 DATA 237,138,33,218,138,34
250 DATA 245,138,33,255,138,34
260 DATA 9,139,62,195,50,29,139
270 DATA 33,116,125,34,30,139
```

```
280 DATA 195,209,138,175,50
290 DATA 166,255,62,195,50,99
300 DATA 255,33,250,250,34,100
310 DATA 255,33,145,125,17,250
320 DATA 250,1,50,0,237,176,195
330 DATA 55,255,33,0,0,34,113
340 DATA 163,33,34,25,34,115
350 DATA 163,62,33,50,120,163
360 DATA 50,123,163,49,255,255
370 DATA 195,48,112
```

* * *

INTO THE EAGLE'S NEST

Набрав этот блок Вы не только станете непобедимым, но еще и получите неограниченное количество боеприпасов и ключей от дверей замка. Запрос в строке 200 относится к неограниченному боекомплекту. Можете при желании от него и отказаться.

Строка 220 - выбор или отказ от непобедимости.

Строка 240 - выбор или отказ от неограниченного количества ключей.

```
1 CLEAR 25599
5 LET t=0: LET w=0
10 FOR f=64000 TO 64037
20 READ a: POKE f,a
30 LET t=t+a*w: LET w=w+1
40 NEXT f
50 IF t<>82517 THEN PRINT "ERROR IN DATA": STOP
100 DATA 33,14,250,17,0,91,1,50
110 DATA 0,237,176,195,0,91
120 DATA 33,255,227,17,255,255,1,0
130 DATA 128,237,184,175,58
140 DATA 32,143,58,176,160,58
150 DATA 64,158,195,0,128
190 POKE 23658,8
200 INPUT "INFINITE AMMO (Y/N)?"; a$
210 IF a$="Y" THEN POKE 64026,50
220 INPUT "INVINCIBLE (Y/N)? ":a$
230 IF a$="Y" THEN POKE 64029,50
240 INPUT "INFINITE KEYS (Y/N)?";a$
250 IF a$="Y" THEN POKE 64032,50
300 PRINT AT 10,0; "START 'INTO THE EAGLE'S NEST' TAPE"
310 LOAD "" CODE
400 POKE 58380,26
410 POKE 58392,250
420 POKE 58695,100
450 RANDOMIZE USR 58368
```

* * *

URIDIUM

Эта программа позволит Вам настроить игру на свой вкус.

Строка 110 - запрос на то, желаете ли Вы иметь бесконечное количество попыток.

120 - свободный пролет сквозь стены и т. п.

130 - игра без противников.

```
1 LET t=0: LET w=0
5 FOR f=64983 TO 65066
10 READ a: POKE f,a
11 LET t=t+a*w: LET w=w+1
12 NEXT f
15 IF t<>397017 THEN PRINT "ERROR IN DATA"
20 DATA 221,33,39,244,17
```

```
25 DATA 125,2,62,255,55
30 DATA 205,86,5,210,215
35 DATA 253,62,48,50,48
40 DATA 245,33,195,0,62
45 DATA 254,34,186,245,50
50 DATA 188,245,33,0,0
55 DATA 34,62,145,195,0
60 DATA 245,33,14,254,17
65 DATA 0,64,1,40,0,237
70 DATA 176,195,0,64,33
75 DATA 255,239,17,255,255
80 DATA 1,0,165,237,184
85 DATA 62,34,58,75,138
90 DATA 62,201,58,86,152
95 DATA 62,195,58,99,138
100 DATA 195,80,253
105 POKE 23658,8
110 INPUT "INFINITE LIVES(Y/N)?"; a$
115 IF a$="Y" THEN POKE 65051,50
120 INPUT "PASS OVER WALLS etc (Y/N)?"; a$
125 IF a$="Y" THEN POKE 65056,50
130 INPUT "NO ALIENS, LAND NOW PRINTED STRAIGHT AWAY (Y/N)?";a$
135 IF a$="Y" THEN POKE 65061,51
150 PRINT AT 10,,3;"START 'URIDIUM' GAME TAPE"
200 RANDOMIZE USR 64983
```

* * *

AMAUROTE

```
1 CLEAR 26590
2 POKE 23658,8
10 PRINT AT 10,5;"START 'AMAUROTE' TAPE"
15 LOAD ""SCREEN$: LOAD ""CODE
20 INPUT "INFINITE MONEY (Y/N)?";a$
25 IF a$="Y" THEN POKE 46381,20
30 INPUT "INFINITE BOMBS (Y/N)?"; a$
35 IF a$="Y" THEN POKE 40615,0
40 INPUT "INVINCIBLE (Y/N)?";a$
45 IF a$="Y" THEN POKE 46312,0
50 RANDOMIZE USR 26600
```

* * *

GAUNTLET

```
1 CLEAR 64999
5 LET t=0: LET w= 0
10 FOR f=65000 TO 65032
15 READ a: POKE f,a
20 LET t=t+a*w: LET w=w+1
25 NEXT f
26 IF t<>64702 THEN PRINT "ERROR IN DATA": STOP
30 DATA 221,33,218,254,17
40 DATA 81,1,62,255,55,205
50 DATA 86,5,48,241,33,1,254
60 DATA 34,57,255,243,195
70 DATA 0,255,62,201,50,82
75 DATA 184, 195, 0, 132
80 PRINT AT 10,5;"START 'GAUNTLET' TAPE"
90 RANDOMIZE USR 65000
```

BETA BASIC

Продолжение. (Начало см. на стр. 3).

Ознакомившись с общими чертами языка программирования БЕТА-БЕЙСИК 3.0, мы теперь можем перейти к подробному рассмотрению его команд и функций.

РАЗДЕЛ 2. КОМАНДЫ

Команды приведены в алфавитном порядке.

1. ALTER <атрибуты> TO атрибуты

Ключевое слово расположено на клавише "A". Команда ALTER позволяет выполнять значительные манипуляции с атрибутами экрана (INK, PAPER, BRIGHT и FLASH) для каждого знакоместа. В своей простейшей форме команда ALTER может изменять атрибуты по всему экрану, не очищая его.

Пример.

```
100 PRINT AT 10,10: "TEST": PAUSE 50: ALTER TO PAPER 1
```

Эта строка изменит цвет PAPER всех символов на экране и сделает его синим. Можно провести по всему экрану установку и какой-либо комбинации атрибутов.

```
ALTER TO PAPER 2,INK 7, FLASH 1
```

Можно производить и выборочную смену атрибутов. Для этого надо перед TO указать какие атрибуты Вы хотите поменять и на какие:

```
ALTER INK 7 TO INK 0
```

В этом примере все, что написано на экране белым цветом INK изменится на черный. Перед вами открывается возможность создавать оригинальные видеоэффекты. Например, Вы можете изобразить что-либо одинаковым цветом INK и PAPER и тогда изображение на экране будет не видно. Теперь командой ALTER Вы меняете INK или PAPER и изображение проявляется перед вами.

Попробуйте сделать вот такую головоломную комбинацию - она тоже будет работать:

```
ALTER INK 3, BRIGHT 1, PAPER 7 TO INK 5, FLASH 1
```

- но затронет только те знакоместа, в которых установлены INK=3,BRIGHT=1 и PAPER =7 одновременно.

Следующая программа продемонстрирует некоторые из эффектов техники применения ALTER. Попробуйте поэкспериментировать с этим оператором. Скорость мигания полей на экране Вы можете изменять в строках 170 и 220.

```
100 LET a=2,b=4
110 FOR I=1 TO 5
120 FOR n=1 TO 16
130 PRINT INK a; PAPER b; "XXXX"; PAPER a; INK b; "OOOO";
140 NEXT n
150 LET c=a, a=b, b=c
160 NEXT I
170 LET t=30
180 ALTER INK a TO INK b: PAUSE t
190 ALTER PAPER a TO INK a: PAUSE t
200 ALTER INK a TO PAPER b: PAUSE t
210 ALTER INK b TO PAPER a: PAUSE t
220 LET t=t-t/10+1
230 GO TO 180
```

2. ALTER <ссылка> TO ссылка

Ключевое слово расположено на клавише "A". Это принципиально иная разновидность команды ALTER. По этой команде программа ищет появление первого значения и заменяет его на второе. Ссылкой здесь может быть имя переменной, число или

строка символов. Например:

```
ALTER a$ TO b$
```

- заменит переменную a$ на b$;

```
ALTER count TO c
```

- заменит переменную count на переменную c, но не затронет переменные accounts, counter и т. п.

```
ALTER 1 TO 23
```

- заменит число 1 на число 23. При этом, что очень важно, изменится также и "невидимое" пятибайтное представление числа, т.е. не только на экране вместо единицы будет изображено число 23, но и в расчетах тоже будет участвовать число 23.

Однако:

```
ALTER 1 TO "23"
```

- заменит число 1 и его пятибайтное представление на пару символов "23" и расчеты в программе будут производиться неправильно, хотя то, что Вы увидите на экране в измененных строках, будет записано вроде бы правильно.

Для всех вышеприведенных примеров по команде ALTER производится поиск по программе, при этом то, что стоит в кавычках, поиску не подлежит (предполагается, что Вы шлете и заменяете название процедуры, имя переменной или число). Но, в то же время, символьная строка будет найдена в программе где угодно, даже внутри кавычек, например:

```
ALTER "break to stop" TO "any key to stop"
```

Сами же кавычки при этом не разыскиваются. Если же вам надо вместо одной из символьных строк использовать имя переменной, то оно должно быть заключено в скобки для того, чтобы команда ALTER понимала, что Вы желаете изменить не имя переменной, а ее содержимое. Например:

```
LET s$ = "execute": ALTER (s$) TO "execute"
```

Если же Вам надо использовать ALTER с символьными строками, содержащими ключевые слова, впечатайте их, воспользовавшись SYMBOL SHIFT/ENTER для того, чтобы принудительно включить курсор "K".

Вы можете использовать ALTER для быстрого удаления чего-либо из программы путем замены на пустую строку, например:

```
ALTER "word" TO ""
```

Вы, наверное знаете (а в "ZX-РЕВЮ-91" мы этот вопрос обсуждали) , что числа требуют большого расхода памяти в БЕЙСИКЕ по сравнению с переменными и выражениями. Это происходит вследствие того, что после символьного выражения числа следует еще его пятибайтный аналог. И нередко в программах производят подмену, например вместо 1 употребляют SGN PI (экономятся 5 байтов) или, скажем вместо любого числа применяют VAL "число" (экономятся 3 байта). Это можно легко сделать с помощью ALTER. Например, для чисел от 1 до 100.

```
1 LET free = MEM()
2 FOR  n=1 TO 100
3 ALTER (n) TO "VAL" + CHR$ 34 + STR$ n + CHR$ 34
4 NEXT  n
5 PRINT "saved "; MEM()+ 28 - free;" bytes"
```

Программа напечатает Вам сколько байтов ей удалось сэкономить. Переменная n в строке 5 стоит в скобках для того, чтобы по команде ALTER изменялось не имя переменной "n", а то число, которое она хранит. Номера строк взяты минимальными - от 1 до 5 и это сделано специально. Дело в том, что при такой работе ALTER возможны сокращения расхода памяти в БЕЙСИК-области и все строки могут "поехать" вниз в адресах памяти, а это недопустимо для циклов FOR... NEXT. Так что нельзя допускать, чтобы до этого цикла в программе могли бы быть какие то строки, способные измениться по ALTER.

Число "28" в строке 5 введено для того, чтобы компенсировать те затраты памяти, которые потребовались на создание нужных в этой процедуре переменных free и n, т.е. чтобы эксперимент был чистым.

И последнее замечание. Будьте очень осторожны при использовании команды ALTER в данной форме. Неаккуратной работой Вы можете легко внести в программу неисправимые изменения. Например, если Вы измените все переменные "apple" на "a", а потом

сообразите, что "а" уже использовалась в программе, то будет поздно. Вы не сможете сделать обратный ход и снова поменять "а" на "apple", поскольку при этом изменятся и те переменные "а", которые и должны быть "а". Можно прежде, чем делать такие замены, убедиться, что "а" в программе не использовалась - например командами REF или LIST REF (см. далее).

3. AUTO <номер строки> <,шаг>

Клавиша - "6".

AUTO - включает режим автоматической нумерации строк, что делает более удобным написание программ. Если значение шага не указано, предполагается, что шаг равен 10. Если при этом не указан и номер строки, с которого начнется автонумерация, то предполагается номер текущей строки (строки, в которой установлен программный курсор) плюс десять.

Режим AUTO отключается, когда номер строки менее 10 или более 9983 или при выдаче любого системного сообщения. Обычно принято выходить из AUTO нажатием и удержанием BREAK продолжительностью более секунды.

Если, находясь в режиме AUTO, Вы хотите выпустить некоторый блок строк, сотрите предложенный программой номер строки и впишите вместо него свой собственный. Тогда следующим номером, предложенным Вам, будет тот, что Вы ввели, плюс величина шага.

Примеры.

AUTO - от текущей строки + 10 с шагом через 10;

AUTO 100 - от строки с номером 100 и с шагом через 10;

AUTO 100,5 - от строки с номером 100 и с шагом через 5.

4. BREAK

Клавиши - CAPS SHIFT + SPACE в неграфическом режиме.

BREAK - не ключевое слово. Обычная команда BREAK из стандартного БЕЙСИКа вполне подходит для большинства приложений, но те, кто программируют в машинных кодах, знают, как часто программа входит в замкнутый цикл и не прерывается нажатием BREAK.

БЕТА-БЕЙСИК имеет резервную BREAK-систему. Если Вы нажмете и удержите SHIFT + SPACE более, чем в течение 1 секунды, эта система поймет, что Вы "зависли" и исполнит BREAK даже если обычная система не работает. Этим методом можно выйти из затруднений, если Вы неосторожно отключили "STOP in INPUT" или "BREAK into program" посредством команды ON ERROR.

Этот же прием выведет Вас из INPUT LINE, EDIT и AUTO.

Примечания:

1) Если Вы "вывалились" в исходное "Синклеровское" сообщение, этот метод Вам не поможет.

2) Программистам в машинных кодах, желающим воспользоваться работой резервной BREAK-системы, не следует отключать прерывания.

Система работает на прерываниях 2-го рода и они должны быть включены постоянно.

5. CLEAR число.

Эта команда должна представлять большой интерес для тех, кто работает с машинным кодом. CLEAR с числом, меньшим чем 767 перемещает указатель RAMTOP вниз на заданное этим числом количество байтов.

Почему 767? В этом числе всего три знака, поэтому Вы его никак не спутает с обычной командой CLEAR, после которой, как известно должен идти пятиразрядный адрес. Кроме того, это число легко реализуется на Ассемблере.

На экран, переменные и на стеки GOSUB, DO-LOOP и PROC эта команда не влияет.

Клавиши определенные пользователем, а также область определения окон (расположенная непосредственно выше RAMTOP) перемещаются вместе с RAMTOP. За этими определениями образуется свободное пространство, простирающееся до начала кода БЕТА-БЕЙСИКа 47070. Обратите внимание: это пространство не заполняется нулями.

Та же команда CLEAR с отрицательным числом передвинет RAMTOP и области определения клавиш пользователя и окон - вверх на заданной число байтов.

Никаких проверок на перекрытие машинного кода БЕТА-БЕЙСИКа не производится, поэтому будьте осторожны и вообще не надо пользоваться этой командой, если Вы ранее не перемещали RAMTOP вниз.

Маленький нюанс. Когда Вы опускаете RAMTOP вниз на сколько-то байтов, образуется свободная область размером столько же байтов. Она образуется непосредственно под машинным кодом БЕТА-БЕЙСИКа и вовсе не обязательно над RAMTOP, поскольку RAMTOP могла указывать в совсем иное место.

Пример позволяет Вам поэкспериментировать с этой командой.

```
100 LET ramtop = 23730
110 PRINT DPEEK (ramtop)
120 CLEAR 100
130 PRINT DPEEK (ramtop)
140 CLEAR -50
150 PRINT DPEEK (ramtop)
```

6. CLOCK число или строка

Клавиша: "C"

Этот оператор позволяет управлять часами, показания которых могут быть показаны в правом верхнем углу экрана. Показания содержат часы, минуты и секунды. Часы - в 24-часовом формате. Может быть задействован режим таймера (будильника). В этом случае в заданное время может быть выдан звуковой сигнал, а может быть включена процедура GOSUB.

Часы работают от прерываний и потому счетчик работает и тогда когда Вы пишете программу и тогда, когда Вы ее запускаете. Единственное, когда часы не работают - тогда, когда идет загрузка/выгрузка, исполняется BEEP или работает теневая периферия типа INTERFACE I.

Параметром, определяющим в каком режиме работают часы, является число, стоящее после оператора. Это число может быть в диапазоне от 0 до 7 и имеет следующее содержание:

Число	Переход GOSUB	Звуковой сигнал	Экран
0	НЕТ	ВЫКЛ	ВЫКЛ
1	НЕТ	ВЫКЛ	ВКЛ
2	НЕТ	ВКЛ	ВЫКЛ
3	НЕТ	ВКЛ	ВКЛ
4	ДА	ВЫКЛ	ВЫКЛ
5	ДА	ВЫКЛ	ВКЛ
6	ДА	ВКЛ	ВЫКЛ
7	ДА	ВКЛ	ВКЛ

Часы начнут работать сразу после загрузки БЕТА-БЕЙСИКа. Начальная установка "00:00:00". Выдать показание на экран можно командой CLOCK 1. Установка конечно не будет соответствовав истинному времени, но поправить дело можно командой:

CLOCK строка , - например:

```
CLOCK "09:29:55"
```

Здесь разряды отделены двоеточием, хотя в этом нет никакой необходимости. При вводе времени для установки, компьютер воспринимает только шесть цифр, а все символы-разделители (кроме символа "a") игнорирует. Если в Вашем вводе будут только пять цифр или меньше, он просто заменит недостающие нулем, например:

```
CLOCK "xyz10"
```

Здесь XYZ будут восприняты как символы - разделители и проигнорированы, а "10" - как установка первых двух разрядов. В результате получится: "10:00:00". Символ "a", о котором мы упоминали, как об исключении, служит для установки таймера: "A06:20" -

установит будильник на 6 часов 20 минут. По достижении этого времени включится звуковой сигнал, если его режим был включен командой CLOCK 2,3,6 или 7.

Можно сделать и так, что в данное время программа выполнит переход GOSUB к назначенной подпрограмме если режим был включен командой CLOCK 4,5,6 или 7. Правда такой переход возможен только в состоянии работающей программы. Если Вы в этот момент выполняете ввод или редактирование. Вашу работу прерывать компьютер не будет.

Подпрограмма, которая вызывается, может быть любой сложности и размера. Это может быть: 10 GO TO 10 - а может быть текстовой редактор или игра. По достижении заданного времени компьютер закончит ту строку, с которой он работает, а потом только сделает переход. Завершение строки может занять определенное время, особенно если это INPUT или PAUSE.

Выбор и назначение подпрограммы, к которой выполняется переход GOSUB делается тоже оператором CLOCK число.

Здесь "число" - это номер строки, к которой делается переход. Это число должно быть в интервале от 8 до 9999. Нижний предел 8 назначен, чтобы отличать это назначение от выбора режима работы, в котором параметр может быть от 0 до 7.

Другой метод ввода подпрограммы для перехода по таймеру - прямой:

```
CLOCK: оператор: оператор:... : RETURN
```

В подпрограмме обработки перехода по таймеру нельзя использовать те же имена переменных, что и в главной программе. Если Вы не хотите, чтобы их содержимое изменялось, Вы можете оформить эту подпрограмму как процедуру и назначить используемые в ней параметры как LOCAL. Если Вы хотите, чтобы эта подпрограмма выполняла сохранение данных, а в главной программе возможно использование RUN или CLEAR, то Вам надо сохранять эти данные посредством POKE в соответствующих областях памяти, неповреждаемых по RUN или CLEAR.

Возможные применения подпрограммы перехода по таймеру могут быть такими:

- Проигрывание музыкальной мелодии в заданное время (разновидность будильника);
- Перестроение экрана;
- Бой часов (чтобы узнать при этом сколько сейчас времени, т.е. сколько раз должны пробить ваши часы можно воспользоваться функцией TIME$ - см. далее)

Вы можете даже изменять масштаб времени с помощью внутренней переменной БЕТА-БЕЙСИКа, размещённой в адресе 56866. Исходно там установлен режим хода 1/50 секунды на каждый ход.

POKE 56866,58 сделает Вам режим 100 секунд в минуте, а POKE 56866,54 - вернет к нормальному режиму.

Специалисты в электронике могут организовать регулярный (скажем, каждый час или каждую минуту) прием данных от внешних устройств, например:

```
8999 STOP
9000 PRINT "Процедура включена"
9010 LET pointer=DPEEK (USR "a"): POKE pointer,IN 127
9020 LET pointer = pointer +1: IF pointer>65535 THEN LET pointer = USR "a" + 2
9030 DPOKE USR "a", pointer: LET z$ = TIME()
9040 LET hours = VAL z$ (1 TO 2), mins = VAL z$ (4 TO 5)
9050 LET mins = mins +1: IF mins = 60 THEN LET hours=hours+1, mins = 0
9060 CLOCK "a" USING$ ("00",hours) + USING$ ("00",mins): RETURN
```

Не запускайте эту программу через RUN, а вместо этого дайте прямую команду:

```
DPOKE USR  "a", USR "a" +2 : CLOCK 9000: CLOCK  5
```

Эта прямая команда проинициализирует указатель (pointer), расположенный в адресах USR "a" и USR "a" + 1. Строка 9000 назначена как строка перехода по таймеру. Команда CLOCK 5 делает этот переход возможным.

Для проверки задайте время срабатывания таймера: CLOCK "AXXXX" - XXXX - установка времени.

Теперь запускайте некоторую программу через RUN. Подпрограмма CLOCK будет активироваться каждую минуту и считывать данные из внешнего порта 127.

Прочитанные данные сохраняются в области графики пользователя UDG в адресах, на которые указывает указатель (pointer). Указатель постоянно наращивается, а при

достижении верхнего предела физической памяти 65535 возвращается в исходное положение. Если у Вас нет внешних устройств, подключенных к внешним портам, то можете организовать работу так, чтобы какое-нибудь другое полезное дело с помощью этой подпрограммы выполнялось через регулярные интервалы времени.

Строки 9040 - 9060 переустанавливают таймер на время, отстоящее на 1 минуту от текущего времени, потом выполняется возврат в главную программу.

7. CLS <номер окна>

Просто команда CLS без указания параметра выполняет очистку текущего окна. (Более подробно см. WINDOW). CLS с параметром, выполняет очистку окна с номером, равным параметру (если конечно это окно было задано). Если же Вы забыли определить это окно, то получите сообщение об ошибке:

```
"Invalid I/O device". (Неверно задано устройство ввода/вывода)
```

Команда CLS 0 - это то же самое, что и команда CLS стандартного БЕЙСИКа, т.е. она выполняет очистку экрана вне зависимости от того, какое окно в настоящий момент является активным.

8. CONTROL CODES (управляющие коды)

Управляющие коды - это специальные символы, употребляемые в командах печати PRINT и PLOT, но сами по себе они не печатаются, а служат для того, чтобы указывать где что печатать. Так что в отличие от обычных символов их нельзя увидеть - они не имеют графического изображения.

В БЕТА БЕЙСИКе есть две группы управляющих кодов. Первая служит для управления курсором и позицией печати по командам PRINT и PLOT, а вторая применяется при управлении специальными блоками экрана, обрабатываемыми по команде GET (см. ниже).

Коды управления курсором.

Коды	Наименование	Область действия
CHR$ 2	Курсор влево	экран
CHR$ 3	Курсор вправо	экран
CHR$ 4	Курсор вниз	экран
CHR$ 5	Курсор вверх	экран
CHR$ 8	Курсор влево	окно
CHR$ 9	Курсор вправо	окно
CHR$ 10	Курсор вниз	окно
CHR$ 11	Курсор вверх	окно
CHR$ 12	DELETE	окно
CHR$ 15	Добавочный ENTER	окно

Разница между кодами с номерами 2...5 и 8...11 в том, что коды, предназначенные для работы в окне, не могут вывести курсор (позицию печати) за пределы текущего окна. Это бывает очень удобным во многих случаях, например при создании текстового редактора.

Коды 2...5 не имеют этого ограничения и применяются, как правило, с командой PLOT. Фактически же PLOT сама конвертирует коды 8...11 в коды 2...5 во время своей работы.

В стандартном БЕЙСИКЕ обработка кода CHR$ 8 имеет ошибку. Так, невозможно перемещением курсора влево поднять его с нижележащих строк на вышележащие, а если он находится на самой верхней строке, то таким перемещением можно его вообще вывести за пределы экрана и войти в распечатку программы.

Здесь эта ошибка исправлена.

Код CHR$ 9 в стандартном БЕЙСИКе тоже обрабатывается с ошибкой и здесь она тоже исправлена.

Стандартный БЕЙСИК распечатывает коды 10, 11, 12 в виде вопросительного знака,

что не очень полезно. Здесь они работают так, как это должно бы быть.

В качестве примера рассмотрим, как управляющие коды, включенные в текстовые строки, позволяют выводить на экран сложные формы через PRINT или PLOT.

```
10 LET a$ = "1235" + CHR$ 8 + CHR$ 10 + "5" + CHR$ 8 + CHR$ 10 +"678" + CHR$ 8 + CHR$ 11 +
     "9"
20 PRINT AT 10,10; a$
30 PAUSE  100: CLS
40 FOR n= 32 TO 255
50 PLOT n, n/2: a$:  NEXT n
```

Особенно полезным это может быть при работе с графикой пользователя - эксперименты проведите сами.

CHR$ 15 работает, как ENTER в том смысле, что позволяет прервать строку в любом месте и продолжить ее набор в новой экранной строке. Обычный ENTER послал бы строку при этом в листинг программы и работа бы с ней закончилась.

Кроме того, CHR$ 15 можно вставлять в текстовые строки, чтобы организовывать печать так, как Вам это нравится. Вводится CHR 15 одновременным нажатием CAPS SHIFT + ENTER.

Коды управления экранными блоками.

Этих кодов два: CHR$ 0, CHR$ 1.

Код CHR$ 0 говорит о том, что за ним следуют восемь байтов, определяющие графический образ элемента экрана.

Код CHR$ 1 говорит о том, что за ним следует байт атрибутов и далее - восемь байтов, определяющие графический образ элемента экрана. Эти управляющие коды совместно с кодами CHR$ 8 и 10 используются командой GET для сохранения блока экрана в качестве строковой переменной. Может быть Вам и не нужна нижеследующая информация, но кому-то она покажется интересной.

Когда команда PLOT встречает символ CHR$ 0, она понимает, что очередные 8 байтов не являются печатными символами, а задают рисунок знакоместа размером 8X8 пикселей, который и должен быть воспроизведен на экране. Это же относится и к команде PRINT, если задан CSIZE, отличный от нуля. (Для нормального экрана следует задавать CSIZE равным восьми.) Кодируется изображение знакоместа по пиксельным строкам абсолютно также, как это делается для символов графики пользователя. Изображается построенный шаблон в текущих установленных цветах INK и PAPER. Все, как с графикой пользователя UDG. Более того, Вы можете создать массив из 9-ти символьных строк, начинающихся с CHR$ 0 и использовать каждый элемент массива, как свой UDG-символ. Понятно, что при этом Вы можете иметь огромные символьные наборы.

CHR$ 1 слегка отличается, т.к. за этим управляющим кодом идет еще один байт, определяющий атрибуты. Поэтому при печати такого шаблона принимаются не текущие цветовые установки, а то, что содержится в этом байте.

Команда GET во время своей работы снимает изображение заданной области экрана, режет его на знакоместа, каждое знакоместо кодирует 8-ью или 9-ью байтами, добавляет перед каждой серией байтов CHR$ 0 или CHR$ 1, вставляет между сериями символы управления курсором и, тем самым, представляет область экрана в виде длинной строки символов, которую и запоминает в какой-то переменной.

Если хотите поэкспериментировать с управляющими кодами, создайте строку и напечатайте ее сразу всю целиком, а не по одному символу.

```
10 CSIZE 8
20 LET a$ = CHR$ 0 + CHR$ 255 + CHR$ 129 + CHR$ 129 + CHR$ 129 +CHR$ 129 + CHR$ 129 + CHR$
     129 + CHR$ 255
30 PRINT a$: PRINT CSIZE 16; a$: PLOT 128,88; a$
```

9. COPY строка COPY массив

Команда имеет очень близкое отношение к команде JOIN. Подробности см. в команде JOIN.

10. CSIZE ширина <, высота>

Клавиша: SHIFT + 8.

CSIZE управляет размером символов при использовании операторов PLOT, PRINT и LIST. Аналогично INK и PAPER эта команда имеет глобальный характер, когда используется в качестве самостоятельного оператора и распространяется только на один оператор, если стоит в качестве элемента в списке PRINT.

Ширина и высота задаются в единицах пикселей. Если Вы не задаете параметр высоты, то по умолчанию высота принимается равной ширине. Нижеприведенный пример показывает символы, имеющие размеры в четыре раза больше, чем стандартные (CSIZE 32). Но Вы можете сделать и символы во весь экран - CSIZE 255,176. Правда, для очень больших символов придется нажимать BREAK, чтобы остановить автоматический скроллинг экрана.

```
10 FOR n=8 TO 32 STEP 8
20 CSIZE n
30 CLS
40 PRINT "CSIZE  "; n
50 LIST
60 NEXT n
70 CSIZE 0
```

Символы больших размеров выполняются пропорциональным увеличением стандартных "Спектрумовских" символов. CSIZE 16 - увеличение в 2 раза, CSIZE 24 - в три раза и т. д. Небольшие изменения в CSIZE могут повлиять на расстояние между символами при печати, не влияя на размеры самих символов.

Попробуйте в приведенном примере в строке 10 удалить оператор STEP 8 и посмотрите, что получится.

CSIZE с параметрами 8,9,10,11 работает с шрифтом 8X8. CSIZE с параметрами 12...19 использует уже шрифт 16X16 и т.д.

Для некоторых параметров возможно, что при печати поля очередного символа будут перекрывать (затирать) рядом лежащий символ. Неплохой идеей борьбы с этим является использование при печати режима OVER 1.

Для вышеприведенного примера, можно получить интересные эффекты, если перед командой LIST дать прямые команды:

```
INVERSE 1: CSIZE 9
```

или

```
CSIZE 32,8
```

или

```
CSIZE 8,32
```

Итак, мы рассмотрели работу крупными символами, но можно поработать и с мелкими. CSIZE 3,8 позволит Вам иметь 85 символов строке, но это не очень зрелищно, разве что если Вы работаете только с малыми буквами и имеете хороший монитор.

CSIZE 4,8 дает 64 символа строке. CSIZE 6,8 и CSIZE 7,1 используют стандартные символы экономят на пробелах между ними. Если Вам надо иметь 40 символов строке, делайте так:

```
OVER 1: CSIZE 6,8
```

После этого, с помощью команды WINDOW уменьшите размер экрана по ширине до 240 пикселей и у Вас будут точно 40 символов в строке.

Подпрограмма, выполняющая печать, отличается большой ухищрённостью. Она может печатать в любой точке экрана (с координацией позиции печати до пикселя), в любом размере, согласует печать с активным в данный момент окном.

Поэтому она работает конечно медленнее, чем стандартная PRINT процедура. Для возврата к стандартной процедуре Вы можете использовать специальную команду CSIZE 0. При этом в качестве активного окна выбирается WINDOW 0 (весь экран).

После загрузки БЕТА БЕЙСИКа исходной является установка CSIZE 0.

Все прочие коды управления печатью такие как TAB, AT, запятая, PRINT, коды управления курсором - работают в соответствии с установленным значением CSIZE. Например, для режима CSIZE 4,8 Вы сможете дать такую команду, как TAB 63. Внутри

операторов PRINT и PLOT, CSIZE используется для создания временных эффектов.

```
10 PRINT CSIZE 8,16:"двойная высота"; CSIZE 8; "нормальная высота"; CSIZE 4,8; "малая
    высота"
20 PLOT CSIZE 32,16;100,100; "AS"
```

При печати символов графики пользователя есть небольшие особенности. Если ширина задана меньше, чем 6 пикселей, печатается только правая часть символа. Вы, конечно, можете подготовить свой набор символов, учитывающий этот факт.

Символы блочной графики обрабатываются как обычные символы - растягиваются, сжимаются и пр. Есть ограничения на печать блоков экрана, снятых по GET. Этот блок, как мы говорили выше, представляет собой строковую переменную, определяющую конструкцию шаблона 8X8 и взаимосвязь между соседними шаблонами.

Поэтому при печати таких блоков желательно использовать задания CSIZE, кратные восьми, в крайнем случае - 4, т.е. это 4,8,16,24 и т.д.

При использовании параметра 4 могут исчезать отдельные пиксели, но иногда для регулярных рисунков результат получается неплохим.

10. DEFAULT переменная = значение <,переменная = значение>...

Клавиша: SHIFT + 2.

DEFAULT в принципе работает, как и LET, но в отличие от него не изменяет значение переменной, если она уже существует.

```
10 LET a=10
20 DEFAULT a=20
30 DEFAULT b=30
40 PRINT a,b
```

Строка 40 напечатает Вам, что a=10, поскольку DEFAULT в строке 20 будет проигнорирован, а b=30, т.к. DEFAULT в строке 30 сработает.

DEFAULT следует понимать как выражение принять значение, если иное не задано. Как и за LET, в БЕТА-БЕЙСИКе за DEFAULT могут идти множественные определения, причем обрабатываются они независимо, одно за другим.

```
100 DEFAULT a$="abedef",zx=123,q=0
```

Этот оператор можно использовать в программе где угодно, но наиболее широкое применение он имеет в определениях процедур для того, чтобы пользователь мог опускать те или иные параметры при вызове процедуры.

11. DEFAULT = устройство

Эта разновидность оператора DEFAULT позволяет задавать по умолчанию устройство ввода/вывода. Она рассчитана на работу с ИНТЕРФЕЙСОМ-1.

Команда позволяет использовать обычные команды SAVE/LOAD/MERGE/VERIFY при работе с микродрайвом, локальной сетью, глобальной сетью через порт RS232.

То есть, благодаря ей, упрощается синтаксис команд в работе с этими видами устройств.

В состоянии поставки БЕТА-БЕЙСИК имеет установку "t", то есть на магнитофон. Примеры прочих установок:

DEFAULT = m	Микродрайв 1
DEFAULT = M1	Микродрайв 1
DEFAULT = M2	Микродрайв 2
DEFAULT = n5	Локальная сеть, станция N5
DEFAULT = B	Порт RS232, канал "B"
DEFAULT = t	Магнитофон

В качестве примера покажем, какие команды могут быть использованы после того, как была проведена установка DEFAULT = m. Все они будут работать с микродрайвом 1.

```
SAVE "test"
SAVE 1; "test"
SAVE "m"; 1; "test"
SAVE 10 TO 100; "test"
LOAD "test"
```

```
VERIFY "test"
MERGE "test"
ERASE "test"
CAT
```

При этом, чтобы использовать микродрайв 2, можно либо задать DEFAULT = m2, а можно и не задавать, а использовать:

```
SAVE 2, "test"
LOAD 2, "test"
ERASE 2, "test"
CAT 2
```

Вы можете вместо номера использовать переменную, например:

```
LET x=2: DEFAULT mx
```

Но нельзя использовать для этой цели строковую переменную.

Даже если в качестве устройства ввода/вывода назначен магнитофон, такие команды как SAVE 1; "abc" или LOAD n,"prog" будут работать с микродрайвом, поскольку заданный параметр номера однозначно указывает на то, что речь не идет о магнитофоне. Текущие параметры установки устройства будут выгружены вместе с кодом БЕТА-БЕЙСИКа, если Вы захотите выгрузить свою программу и БЕТА-БЕЙСИК вместе с ней.

Примечание: БЕТА-БЕЙСИК разрешает в командах, связанных с микродрайвом использовать запятую "," вместо точки с запятой ";".

12. DEF KEY односимвольная строка; строка

или

DEF KEY односимвольная строка; оператор: оператор: ...

Клавиша: 1 (та же, что и DEF FN).

См.также LIST DEF KEY

БЕТА-БЕЙСИК позволяет присвоить любой цифровой или буквенной клавише значение символьной строки или программных строк. При этом символы могут вводиться в компьютер, а могут оставаться в нижней части экрана (в области редактирования) до нажатия ENTER.

Последний случай реализуется следующим образом: надо сделать так, чтобы последним символом Вашей строки стояло двоеточие ":" или чтобы последним оператором программной строки стоял оператор ":". Попробуйте:

```
DEF KEY  "1"; "HELLO: "
```

Теперь нажмите совместно SYMBOL SHIFT и SPACE. Курсор изменится и станет мигающей звездочкой. Если Вы теперь нажмете клавишу "1", в нижней части экрана появится надпись HELLO. Поскольку никакие другие клавиши не были переопределены, при их нажатии Вы получите их нормальные значения.

```
DEF KEY "a":PRINT "Goodbye"
```

В этом примере часть программной строки, следующая после DEF KEY "a", присваивается клавише "a" (или "A", что одно и то же). Эта строка не исполняется после ее набора. После же того, как Вы нажмете SYMB SHIFT + SPACE и затем нажмете "a", она будет введена и исполнена, поскольку она не заканчивается двоеточием.

```
DEF KEY "a"; "10 REM hello"
```

Такая строка после нажатия на клавишу "a" будет реально введена в листинг программы после того, как пройдет проверку на синтаксис.

Клавише можно сделать переприсвоение в любое время. Старое значение при этом будет переписано. Если присвоить клавише пустую строку или если после задания клавиши нет ни одного оператора, клавиша не будет иметь определения. Оператор DEF KEY ERASE уничтожит все сделанные определения, в том числе и те, которые размещены выше RAMTOP и, тем самым, защищены даже от NEW. Процедура SAVE в загрузчике БЕТА БЕЙСИКа выгрузит все определения клавиш вместе с машиннокодовой частью БЕТА-БЕЙСИКа (она производит выгрузку кода от RAMTOP до конца БЕТА-БЕЙСИКа).

Чтобы просмотреть все определения клавиш, пользуйтесь командой LIST DEF KEY. Если желаете подредактировать определение, присвоенное клавише, то можете выполнить

следующий прием. Наберите номер строки, затем нажмите номер желаемой клавиши и Вы получите редактируемую строку, которую Вы сможете оформить в оператор DEF KEY. RAMTOP автоматически понижается, когда Вы задаете клавиши. Если Вы воспользуетесь обычным CLEAR для изменения RAMTOP, то вполне может быть, что Ваши определения клавиш пропадут. С другой стороны, БЕТА-БЕЙСИК имеет другой вариант CLEAR (см. выше), с помощью которого можно без риска для определений клавиш выделять пространство для своего машинного кода выше уровня RAMTOP.

13. DEF PROC имя процедуры <параметр><,REF параметр>...

или

DEF PROC имя процедуры <DATA>

Клавиша: 1 (та же, что и DEF FN).

См. также PROC, END PROC, LOCAL, DEFAULT, LIST PROC, REF,ITEM.

Оператор DEF PROC открывает определение процедуры. Он должен быть первым оператором в программной строке (разрешаются только ведущие пробелы или предшествующие управляющие цветовые коды). Имя процедуры должно обязательно начинаться с буквы, а заканчиваться может пробелом, двоеточием, REF, ENTER или DATA. Имя процедуры может быть записано почти любыми символами, за исключением пробела, но желательно, чтобы Вы использовали буквы, цифры, знак подчеркивания "_", и, пожалуй символы "#" и "$". Буквы верхнего и нижнего регистров - эквивалентны. Процедура может иметь то же имя, что и переменная и путаница не произойдет.

За именем процедуры может идти список параметров или ключевое слово DATA. Параметры, заданные в определении процедуры, называются формальными параметрами. Это имена переменных. Когда процедура будет вызвана, то все имеющиеся в ней переменные с именами, совпадающими с формальными параметрами, будут защищены и только потом им будут присвоены значения фактических параметров, присутствующих в вызове. Когда же перед именем формального параметра стоит REF, все происходит почти так же, но в добавок по окончании работы процедуры значение формального параметра будет присвоено соответствующему фактическому параметру. В процедуре могут участвовать и массивы, но они обязательно должны быть оформлены как ссылки, то есть перед именем массива в определении процедуры обязательно должно стоять REF.

Когда вместо списка параметров стоит ключевое слово DATA, никаких переприсвоений не делается, а список фактических параметров при вызове процедуры может быть принят с использованием оператора READ и функции ITEM.

14. DELETE <номер строки> TO <номер строки>.

Клавиша 7 (та же, что и ERASE).

Команда удаляет все строки в указанном блоке программы. Если номер начальной строки опушен, предполагается минимальный, отличный от нуля. Если опущен номер конечной строки удаления, принимается номер последней строки программы.

Примеры.

DELETE TO 100 - удаляет все строки после нулевой и до строки 100 включительно;

DELETE 100 TO - удаляет строку 100 и все последующие;

DELETE 100 TO 100 - удаляет только строку 100;

DELETE 0 TO 0 - удаляет только нулевую строку;

DELETE TO - удаляет всю программу, за исключением нулевой строки.

Последний пример отличается от NEW тем, что не вычищает программные переменные. Все строки, явно указанные в операторе, должны реально существовать, иначе Вы получите сообщение об ошибке:

```
U "No such line" (нет такой строки).
```

DELETE можно включать в текст программы, но с некоторыми предосторожностями. Когда вышележащие строки программы удаляются оператором DELETE, расположенным в подпрограмме, процедуре, в цикле FOR...NEXT или DO...LOOP, программа обычно прекращает нормальную работу, поскольку запомненный адрес возврата уже не

соответствует реальным новым адресам строк. Если же оператор DELETE применяется для того, чтобы удалить самого себя из программы, он должен быть последним оператором в строке.

Возможное применение DELETE в программе - это удаление строк DATA после того, как они уже были прочитаны для того, чтобы освободить память для программных переменных, поскольку числа в строках DATA очень сильно расходуют память - по крайней мере по 8 байтов на каждое число. Программа может также удалять часть строк для того, чтобы подзагрузкой выполнить MERGE нового блока на освободившееся место.

15. DELETE имя массива <пределы>

или

DELETE строка <пределы>

Клавиша 7 (та же, что и ERASE)

Кроме удаления программных строк, DELETE может удалять массивы полностью или частично и символьные строки.

```
10 LET a$ = "123456789"
20 DELETE a$ (4 TO 7)
30 PRINT a$: REM Prints "12389"
40 DELETE a$
50 PRINT a$: REM переменная не найдена
```

Когда удаляется строковая переменная, то она перестает существовать полностью, а не просто становится пустой строкой. Команда работает одинаково для строк и для массивов. Создайте массив для эксперимента и попробуйте:

```
10 DIM a$ (10,4)
20 FOR n=1 TO 10
30 LET a$(n)=CHR$(64+n)+"xxx"
40 NEXT n
50 FOR n=1 TO length (1,"a$")
60 PRINT a$ (n)
70 NEXT n
```

В строке 50 использована функция LENGTH вместо числа 10 потому, что количество элементов в массиве будет изменяться. Теперь добавьте дополнительные строки:

```
45 DELETE  a$ (3)
45 DELETE  a$ (3 TO)
45 DELETE  a$ (TO 4)
45 DELETE  a$
```

Опять запустите программу командой RUN и посмотрите, какие элементы будут удаляться. В числовом массиве команда DELETE a(6) удалит шестой элемент, если массив одномерный или целую строку элементов, если массив двумерный.

DELETE a(3 TO 8) удалит "вырезку" из элементов одномерного массива или группу строк из двумерного. (В числовом двумерном массиве количество рядов задаете первым числом.)

Продолжение следует.

МИХАИЛЕНКО В.С.

ЗАЩИТА ПРОГРАММ

Сегодня мы продолжаем разговор о приемах защиты программ от просмотра. Начало статьи см. в "ZX-РЕВЮ" 1-2, 1992 г., стр. 9-16.

2.3.4. Управляющие коды, используемые для защиты от просмотра.

Естественно, что того описания которое приведено в разделе 2.3.3, недостаточно для полноценного использования управляющих кодов. Здесь мы более подробно рассмотрим их применение для защиты от листинга, изучая программы, взломанные известными хаккерами, а также просматривая наиболее удачные программные приемы фирменных программ.

Итак по порядку.

Код N7 EDIT не используется для целей защиты. Это управляющий код, который создает программа ввода с клавиатуры при нажатии клавиш "CAPS SHIFT" и "1" для вызова строки в область редактирования. Эта область изменяется в соответствии с размерами строки.

N8 BACKSPACE. Этот код, означающий в переводе с английского "Курсор назад" достаточно широко используется для скрытия некоторых частей строки Бейсика, для накладывания частей друг на друга с целью дезинформации и для создания необычных эффектов, один из которых был нами рассмотрен в примерах (раздел 2.3.2).

Естественно, можно использовать этот код сразу для нескольких целей одновременно. Рассмотрим теоретические способы использования BACKSPACE для скрытия информации и исполнения накладок (с практическими приемами Вы будете ознакомлены в следующей статье).

Предположим, первой строкой в Вашей программе идет строка, выполняющая реальные действия, например осуществляющая запуск программы в машинных кодах, встроенной в Бейсик (о том как сделать такую программу читайте в Главе 1. Эта программа осуществляет реальную загрузку и только она может правильно загрузить и запустить блок кодов с магнитной ленты.

Чтобы ввести пользователя в заблуждение, текст этой строки скрывают, а в следующих за ней строках ставят ловушки (естественно эти строки не скрывают - пусть все смотрят).

Ловушки могут быть самые разные от продолжения программы на Бейсике, создающей видимость загрузки до запуска программы в кодах в таком месте, где либо коды умышленно путаются с целью зависания компьютера, либо с отправкой на команду АССЕМБЛЕРа JP 0000, аналогичной RANDOMIZE USR 0, что обеспечивает перезапуск компьютерной системы). Таким образом, вся сложность подобной системы защиты сводится к тому, чтобы освоить создание невидимой строки.

Делается это следующим образом: пишется нормальная строка на Бейсике, после этого она зануляется, чтобы её невозможно было вызвать для редактирования, а потом "сжимается" до нуля влево с помощью символов BACKSPACE.

* * *

Примечание: ещё больше затруднить поиск точного адреса старта Вашей программы в кодах можно используя измененный вариант числа управляющим кодом 14, который описан ниже, а кроме этого рекомендуется искажать подлинный смысл операторов Бейсика с использованием методов, описанных в разделе "Новейшие достижения защиты".

* * *

Более подробно как это сделано показано на рис. 1:

```
 MM NN|<-|<-|<-|<-|<-|<-|<-|<-|
|<-|<-|<-|<-|<-|<-|<-|<-|<-|<-|
|<-|<-|<-| RANDOMIZE USR ABCDE
```

РИС. 1

где MM - номер строки NN - длина строки ABCDE - номер ячейки памяти, с которой осуществляется запуск программы в кодах.

После Бейсик строки мы размещаем ровно столько символов BACKSPACE чтобы текст строки был скрыт из виду т.е. каждому символу строки соответствует символ BACKSPACE.

Сокрытие лишь части строки из виду является разновидностью данного метода и используется в аналогичных целях. В частности, в широко распространенной программе COMMANDO (фирма ELITE) - оно используется следующим образом:

```
10 CLEAR  40000
20 LOAD  ""  CODE
30 RANDOMIZE USR (неверный, т.е. ложный шаг, с которого якобы начинается программа в кодах.)
```

Достаточно хитро спрятан действительный адрес, по которому происходит запуск программы в кодах. Он размещен в строке 20 после команды загрузки LOAD "" CODE и "сделан невидимым" с помощью символов BACKSPACE. Для тех, кто интересуется более подробно, как это сделано, рекомендую составить программу для непосредственного просмотра ячеек памяти Бейсик-программы (дампинга памяти), разместив ее в конце программы:

```
9997 FOR I=23755 TO 30000
9998 PRINT PEEK I; " ";CHR PEEK I; " "; I
9999 NEXT I
```

Запускается эта программа командой GO TO 9997, а в случае остановки ее из-за невозможности распечатать тот или иной символ или по другим причинам, необходимо набрать с клавиатуры NEXT I и дампинг продолжится. Этот метод рекомендуется применять для просмотра содержимого любых программ, т.к. он позволяет получить истинную картину Бейсика.

Коды N9, N10 , N11 - RIGHTSPACE, DOWNSPACE и UPSPACE для защиты программ не используются, они выделены в отдельную группу т.к. генерируются программой ввода с клавиатуры для передвижения курсора в заданном направлении:

"CAPS SHIFT" +8 - -> RIGHTSPACE
"CAPS SHIFT" +6 - -> DOWNSPACE
"CAPS SHIFT" +7 - -> UPSPACE

Коды N12, N13 - DELETE и ENTER для защиты программ не используются. N14: - этот управляющий код предшествует числам в программах. Как известно, в "СПЕКТРУМЕ" при программировании на Бейсике, обычные числа являются наибольшими расточителями памяти. Это происходит потому, что фактически числа записываются в память дважды - первый раз записано число в том виде, в каком оно печатается на экране, а второй раз записано истинное значение числа, т.е. то, которое обрабатывается интерпретатором. Свидетельством того, что началась запись числа для интерпретатора и является управляющий код 14.

Введем, например, строку:

```
10 PLOT 10,9
```

и посмотрим, каким образом она запишется в память. Выглядит она так, как показано на рис. 2:

0 10	18 0	248	49 48	14 0 0 10 0 0	44	57	14 0 0 9 0 0	13
10	18	PLOT	1	10	,	9	9	
номер строки	длина строки			число 10			число 9	ENTER

Рис 2.

Как видите, текст, который хранится в памяти, отличается от того, что изображается на экране. После последней цифры каждого числа, выступающего в тексте строки как параметр функции PLOT, интерпретатор выделил 6 байтов памяти и поместил там символ с кодом 14, а за ним - 5 байтов, в которых записано значение этого числа (число записано способом, понятным интерпретатору. Для желающих более подробно изучить способы представления чисел в "СПЕКТРУМЕ" рекомендую обратиться к методической разработке "ИНФОРКОМа" "Первые шаги в машинных кодах". Это ускоряет в определенной мере выполнение программ на Бейсике, т.к. во время выполнения интерпретатор не должен каждый раз переводить числа из алфавитно-цифрового представления (просто последовательность цифр) в 5-ти байтовое представление, пригодное для вычислений на встроенном в ПЗУ калькуляторе "СПЕКТРУМА". Готовое значение выбирается из памяти после управляющего кода 14.

Рассмотрим небольшой пример конкретной защиты программ. Во многих программах загрузчиках присутствует такая строка:

```
0 RANDOMIZE USR 0: REM...
```

На первый взгляд эта строка после запуска должна перезапустить компьютер и, соответственно, очистить всю память "СПЕКТРУМА", но этого не происходит. После более внимательного рассмотрения (с помощью PEEK-программы, которая была предложена ранее) оказывается, что после USR 0 и символа 14 нет 5 нулей, а именно так записалось бы число 0 в пятибайтной форме. Эти значения были умышленно изменены, чтобы сбить Вас с толку.

Но после управляющего символа 14 мы видим не нулевую комбинацию, а определенную последовательность, например: 0,0,218,92,0, что равнозначно числу 23770, т.е. практически функция RANDOMIZE USR не осуществит переход по адресу 0, а делает его именно на адрес 23770, а это как раз адрес байта, находящегося сразу после инструкции REM, где размещена программа в машинных кодах (о том, как разместить её там, мы писали в Главе 1).

N 15 - Этот код "СПЕКТРУМОМ" в стандартном БЕЙСИКе не используется. Использование его в других языках оговаривается в их описаниях, например в БЕТА-БЕЙСИКе 3.0.

N 16 - код управления цветом символов - INK CTRL. После этого символа обязательно наличие одного байта, уточняющего о каком цвете идет речь. Цифровая шкала цветов совпадает со стандартной шкалой цветов СПЕКТРУМА:

0-черный
1-синий
2-красный
3-пурпурный
4-зеленый
5-голубой
6-желтый
7-белый

(В разделе "Секреты ПЗУ" в прошлом году на стр. 148 "ИНФОРКОМ" указывал, что существуют также установки цвета с параметром 8 ("прозрачный") и 9 ("контрастный").

Если после этого управляющего кода будет находиться байт, содержащий значение, не совпадающее со значением одного из цветов, то это вызовет остановку компьютера с выдачей сообщения об ошибке INVALID COLOR (неверный цвет).

Создание системы команд, вызывающих остановку компьютера с выдачей сообщения об ошибке очень часто используется при защите Бейсик-программ от просмотра. В

частности, если мы используем этот метод совместно с методом зануления всех строк то это обеспечит полную защиту листинга, а кроме этого мы сможем оставлять свои сообщения в программах и закрывать доступ к ним (естественно, делать это стоит только на программах, созданных Вами лично).

Рассмотрим более подробно организацию строки Бейсика при использовании управляющего кода INK CONTROL. Как Вам уже вероятно известно, печать всех сообщений на экране "СПЕКТРУМА" осуществляет специальная процедура, встроенная в ПЗУ. Когда эта процедура встречает управляющий код 16, она анализирует следующий за ним байт и принимает его за числовое значение цвета INK, конечно если оно лежит в допустимых пределах.

После того, как код принят, все последующие символы на экране печатаются с цветом INK, соответствующим значению, стоящему после управляющего кода INK CONTROL. В строках Бейсика применение управляющих кодов позволяет экономить память, что достаточно важно в программах и приобретает особую актуальность в загрузчиках.

Рассмотрим, откуда берется эта экономия. Традиционная форма записи Бейсик-строки: INK 0.

Другой вариант - введение в строку управляющего кода INK CONTROL и за ним символа "0". Правда, набрать на клавиатуре эту комбинацию не так просто. О том, как это осуществить практически, смотрите ниже.

И в том и в другом варианте расходуется "приблизительно" одинаковое количество байтов памяти за исключением того, что интерпретатор Бейсика переводит число 0 в специальную 5-ти байтную форму, с которой он и оперирует. То есть фактически при подаче команды INK 0, расходуется не 2 байта памяти а 8:

```
INK 0 14 0 0 0 0 0 ,
```

т.е. в 4 раза больше. Комбинация же:

```
INK CONTROL 0
```

расходует всего 2 байта, поскольку здесь число 0 представлено в своем нормальном виде. Даже метод, использующий комбинацию: INK NOT PI, (что фактически аналогично INK 0) расходует 3 байта памяти, т.е. еще раз убеждаемся, что использование управляющих кодов является самым экономичным из рассмотренных методов.

Еще более существенная экономия места в памяти достигается в том случае, когда управляющие коды используются в строке оператора Бейсика PRINT вместо вставного INK. Это объясняется тем, что в этом случае приходится учитывать символ ";" (точка с запятой), которым необходимо "окаймлять" с двух сторон команду INK.

```
10 PRINT;INK 0; "ZX SPECTRUM"
```

В данном случае под вставку команды INK 0 расходуется 10 байтов:

;	INK	0	14	0	0	0	0	0	;
1	2	3	4	5	6	7	8	9	10

В случае же употребления управляющего кода INK CONTROL истратим лишь 2 байта. Фактически это применение может использоваться не только для защиты программ, но и для элементарной экономии памяти, в частности, очень существенной эта экономия оказалась в программе "MONOPOLY" (аналог известной во всем мире настольной игры), где управляющие коды используются при распечатки сообщений в операторе PRINT.

Практически используя этот и другие приемы, программистам удалось создать великолепный экземпляр игры, в которую можно играть как в одиночку, так и в паре с соперником (под игрой в одиночку я понимаю поединок со "СПЕКТРУМом", когда компьютер выступает вместо второго игрока). Этот код дает Вам возможности не только экономить память. Но и создавать эффекты, достигнуть которые иными путями будет просто невозможно. В частности, хотите ли вы, чтобы при загрузке Вашей программы после ключевого слова PROGRAM текст названия Вашей программы печатается иным (т.е. таким, который Вы заранее зададите) цветом? Если да, то и об этом поговорим на страницах данных статей.

17 управляющий код PAPER CONTROL используется точно так же, как и предыдущий

код INK CONTROL, только в данном случае числовое значение следующее после этого кода изменяет цвет не символа, а фона. В большинстве случаев управляющие коды INK CONTROL и PAPER CONTROL используются вместе для достижения каких либо эффектов. Именно совместное их использование создает наибольшую выразительность и улучшает читаемость информации, а также позволяет достигнуть наилучших цветовых эффектов. Рассмотрим более подробно теоретические аспекты совместного использования управляющих кодов INK CONTROL и PAPER CONTROL на базе ПРИМЕРА 1. (см. раздел 2.3.2).

Вкратце напомню, что тогда мы просто изучали эффекты, возникающие при использовании некоторых управляющих кодов без какой-либо предварительной теоретической проработки. Применение этих символов аналогично часто применяемым операторам изменения цвета INK и PAPER на базе оператора PRINT. Имеются, правда, существенные отличия. Первое - это существенная экономия оперативной памяти (в некоторых случаях этот объем удается сократить в пять раз). И второе отличие состоит в том, что встроенный в PRINT оператор INK или PAPER изменит цвет символов или фона лишь у текста, который должен распечатать PRINT, в то время как соответствующие управляющие символы INK CONTROL и PAPER CONTROL изменяет цвет всего следующего за ними текста. В частности, для эксперимента Вы можете вставить эти управляющие коды в один из фрагментов листинга программы способом, который будет описан в следующей статье (Раздел 2.3.5. "Практическое применение управляющих кодов для защиты). В целях защиты это может применяться, например в тех случаях, когда Вам нужно сделать текст программы невидимым), т.е. действие управляющих кодов INK CONTROL и PAPER CONTROL аналогично действию постоянных операторов INK и PAPER.

Управляющий код 18 задает или отменяет мигание - FLASH CONTROL. После этого управляющего символа следует байт параметра. Кодировка аналогична стандартной кодировке, т.е. она может иметь значение 0,1 или 8. FLASH CONTROL вызывает мерцание всех следующих после этого управляющего кода символов. FLASH CONTROL 8 оставляет в позициях символов прежние ранее установленные значения FLASH.

FLASH CONTROL 0 уничтожает действие предыдущих операторов FLASH 1 и FLASH 8, поэтому все индицируемые после этого управляющего кода символы не мерцают. Учтите, что FLASH CONTROL 1 заставляет мерцать позицию всего символа (8x8), даже в том случае, если высвечивается только одна точка.

Использование этого управляющего кода Вы могли наблюдать при загрузке программы-копировщика "COPY-COPY" в тот момент, когда на экране появляется заголовок "PROGRAM", а после него следуют два слова программы, попеременно мерцающие. Рассмотрим теоретические аспекты применения управляющего кода FLASH CONTROL. Его применение, как и других аналогичных управляющих символов, сопровождается значительной экономией памяти. В некоторых случаях за счет использования FLASH CONTROL в сравнении с FLASH удается "выиграть" до 8 байтов памяти, что при достаточно частом использовании приносит ощутимую экономию.

Для того, чтобы заставить мерцать все символы. Вам необходимо лишь установить управляющий код перед первым из данных символов:

```
FLASH CONTROL 1 <текст программы>.
```

Если же Вы желаете, чтобы мерцал лишь небольшой отрезок текста (именно это необходимо в большинстве случаев), то Вам необходимо перед этим отрезком включить мерцание, а после него - выключить:

```
<текст программы> FLASH CONTROL 1 <отрезок текста> FLASH CONTROL 0 <текст программы>.
```

После ввода вышеприведенных кодов будет мерцать лишь <отрезок текста>.

Ничем не поддержанное использование управляющего кода FLASH CONTROL не принесет эффективности в защиту Вашей программы от несанкционированного доступа. Но применение этого кода может озадачить малоопытного "хаккера", который столкнется с новым для себя приемом программирования.

Управляющий код N19 - BRIGHT CONTROL - индицирует следующие за ним символы более ярко. Это достаточно часто используется в игровых программах, но имеет очень слабую перспективу в применении для защиты программ.

После управляющего кода BRIGHT CONTROL следует байт параметра, который

соответственно придает или отменяет более яркую окраску в зависимости от своего значения:

BRIGHT CONTROL 1 дает более яркую окраску символам, выводимым после этого оператора.

BRIGHT CONTROL 8 указывает, что яркость символов в данном знакоместе должна остаться такой, какой она в нем была и ранее.

BRIGHT CONTROL 0 устраняет действие BRIGHT 1 и BRIGHT 8, и далее символы индицируются с нормальной яркостью.

Теоретические аспекты применения BRIGHT CONTROL полностью аналогичны использованию управляющего кода FLASH CONTROL.

Управляющий код N20 - INVERSE CONTROL меняет местами цвета в позиции символа, т.е. цвет INK становится цветом PAPER и наоборот. После управляющего кода INVERSE CONTROL следует байт, определяющий применение данного управляющего символа. INVERSE CONTROL 1 меняет местами цвета INK и PAPER у всех последующих символов, индицируемых на экране после этого кода.

INVERSE CONTROL 0,соответственно, возвращает первоначальные установки.

Этот управляющий символ также как и BRIGHT CONTROL почти не применяется для защиты программ. Теоретические аспекты применения INVERSE CONTROL аналогичны применению FLASH CONTROL (см. выше).

Управляющий код N21 OVER CONTROL используется тогда, когда необходимо индицировать символ, не уничтожая символа, который уже находился в этой позиции. В зависимости от следующего после OVER CONTROL байта, т.е. того значения, которое находится там, наполнение либо осуществляется, либо нет:

OVER CONTROL 0, который действует по умолчанию, уничтожает в индицируемой позиции ранее индицированный символ.

OVER CONTROL 1 индицирует символ, не уничтожая то, что находится в данном, и, таким образом, получается комбинация символов.

Использование этого управляющего кода может применяться для защиты листинга программ, в частности, совместное его использование с управляющими кодами

BACK SPACE, AT CONTROL, TAB CONTROL, позволит осуществить наложение символов таким образом, что текст программы станет полностью нечитаемым.

Для примера рассмотрим теоретические аспекты совместного применения управляющих символов OVER CONTROL и BACKSPACE для создания готического шрифта. Если на алфавитные символы накладывать какой-либо простой символ, например "/", (наклонная черта), то мы можем получить подобие готического шрифта. Для того, чтобы осуществить наложение, нам необходимо напечатать сам символ, включить режим совмещения, сдвинуть курсор на одну позицию влево управляющим кодом BACKSPACE и после этого напечатать поверх этого символа наклонную черту.

Последовательность действий:

```
B BACKSPACE OVER CNTR 1 /
```

Для пояснения приведем алгоритм на Бейсике, аналогичный вышеприведенной строке. Введем строку:

```
PRINT "B"; CHR 8;OVER 1; "/"
```

после чего мы увидим, как наклонная черта перечеркивает "B".

Здесь был рассмотрен пример, позволяющий украсить шрифт. Однако, нетрудно видеть, что можно умышленно накладывать некоторые символы один на другой так, чтобы исключить возможность прочтения листинга Вашей программы.

Код N22 (AT CONTROL) служит для печати сообщений в заданном знакоместе экрана. Компьютер получает информацию о том, в каком месте осуществлять печать символов после анализа двух байтов, следующих за рассматриваемым управляющим кодом.

Первый байт может быть в пределах от 0 до 21 и указывает номер строки, в которой будут индицироваться данные. Второй байт может лежать в пределах от 0 до 31 и указывает номер колонки, начиная с которой будут индицироваться данные. Неправильное задание данных значений вызовет останов компьютера по ошибке. (Под неправильным применением

в данном случае подразумевается выход значений за допустимые пределы). Управляющий код AT CONTROL является одним из наиболее употребительных и применяется для защиты не реже, чем INK CONTROL и PAPER CONTROL. Возможности его применения очень широки и здесь все будет зависеть от Вашей фантазии.

Рассмотрим теоретические аспекты на конкретных примерах, а к конкретному их применению вернемся позже. Например, Вы хотите, чтобы после остановки Вашей программы по команде BREAK и попытке листинга на экране появлялось заранее заданное Вами сообщение. Причем появляться оно будет в той позиции экрана, в которой Вы желаете. Если хотите, то можете создать комбинацию, которая не будет выводить никаких сопроводительных сообщений - т.е. на экране не будет номера строки и других Бейсиковских атрибутов.

Для достижения подобного эффекта нам необходимо:

Во-первых, уничтожить Бейсиковские атрибуты, используя управляющий код BACKSPACE.

Во-вторых, задать место печати сообщения в необходимом нам месте экрана, используя управляющий код AT CONTROL.

В-третьих изменить цвет INK и PAPER таким образом, чтобы дальнейший листинг программы не был бы виден.

Схема Бейсик-строки, позволяющей достигнуть этого:

```
REM |<-|<-|<-|<-|<-|<-|AT CNTR   |
    |row | col |<текст сообщения>|
    | INK CNTR |0|PAPER CNTR | 0 |
```

Здесь:

<- - упр. код BACKSPACE;

row - номер строки, в которой должно печататься Ваше сообщение;

col - номер колонки, с которой начнется Ваше сообщение хаккеру, взламывающему Вашу программу;

После того, как нам удастся создать эту строку практически, желаемый эффект будет достигнут. Подобное использование AT CNTR практически аналогично использованию AT совместно с PRINT. Но несмотря на это, применение управляющего кода позволяет получать эффекты, достижение которых иными методами было бы невозможно. Вышеприведенный пример является этому подтверждением.

Рассмотрим еще одно любопытное применение управляющих кодов. Это касается использования их в заголовке программ. Многие из Вас, вероятно наблюдали в некоторых программах появление названия программы (следующего после ключевого слова PROGRAM), высвечивающееся не в обычно принятом для этих целей месте, т.е. после слова PROGRAM, а в других местах, в частности, в центре экрана. Кроме этого, в некоторых случаях можно наблюдать появление заголовка вообще без слова PROGRAM (в этом случае название, как правило, печатается с начала строки). Все это достигается с помощью использования управляющего символа AT CONTROL.

Как Вам известно, хэдер (английское название заголовка) состоит из 17 байт. Когда мы загружаем программу, то сначала идет короткий хэдер, а после него непосредственно загружается программа. Хэдер содержит информацию о типе программы(BASIC, CODE и т.д.), о длине программы, о месте размещения программы в памяти компьютера и другие подробности, различные для каждого типа загружающихся программ). Под название программы отводится 10 байтов. Этим объясняется тот факт, что мы не можем записать на ленту программу с количеством символов в названии большим, чем 10.

А теперь давайте представим ситуацию, когда мы вместо первых трех байтов в хэдере введем управляющий код AT CONTROL (возможно введение и любого другого управляющего символа) с соответствующими значениями столбца и строки, а остальные семь байтов у нас будет занимать название программы. В результате этой операции мы получим возможность распечатывать название программы в любом месте экрана. Если же мы будем использовать другой управляющий код, то в зависимости от того, какими

функциями он управляет, будем получать соответствующие изменения. Модель данной строки:

AT CNTR	row	col							
1	2	3	4	5	6	7	8	9	10

Первые три байта занимает управляющий символ AT CONTROL со своими параметрами управлений номером строки и столбца, а байты 4-10 отводятся под название текста программы. Как видим, использование AT CONTROL несколько ограничивает наши возможности - в заголовке остается лишь семь свободных байтов для записи названия, но зато это создает неповторимый эффект. Кроме того, при удачном размещении в оставшихся 7 байтах хэдера ключевых слов Бейсика, Вам может удаться эффект создания полноценного названия, а в некоторых случаях общий объем печатаемого текста будет даже превышать 10 знакомест.

Код N23 (TAB CONTROL) используется для задания позиции данного, либо следующего столбца, начиная с которого в текущей строке печатаются указанные данные. После TAB CONTROL следует один байт значение которого должно лежать в пределах от 0 до 31.

Основные аспекты использования TAB CONTROL для защиты аналогичны подобным моментам для AT, с той разницей, что TAB CONTROL со своим параметром занимает всего 2 байта, в то время как AT CONTROL занимает 3 байта.

Это обстоятельство определяет, преимущественное отношение к TAB CONTROL в тех случаях, когда особенно важна экономия памяти. В частности, использование этого кода в хэдере дает выигрыш в 1 байт, правда несколько оскуднеет эффект применения.

* * *

Мы рассмотрели использование всех, кроме одного, управляющих кодов. Не рассмотренный код COMMA CONTROL оставлен на самый конец повествования, ввиду того, что понять его работу лучше, если знаешь уже теоретические основы применения AT CNTR и TAB CNTR.

Итак:

Код 6 - управляющий код COMMA CONTROL (или управляющая запятая оператора PRINT) используется для индицирования элементов, следующих после этого управляющего кода на смещённых на пол экрана.

Более подробно алгоритм действия COMMA CONTROL можно понять на примере использования запятой в Бейсиковском операторе PRINT. Если в операторе PRINT элементы данных разделены запятыми, то они начинают индицироваться или с начала экрана или с его середины.

Наиболее любопытным аспектом при использовании COMMA CONTROL является тот факт, что это не простое табулирование 16 символов, т.к. при этом очищаются все 16 знакомест, по которым осуществлялась табуляция. Это значит, что COMMA CONTROL можно использовать для стирания некоторых надписей на экране, особенно в тех случаях когда необходима особая экономия памяти. Одним из этих случаев является применение COMMA CONTROL в хэдере. Именно с использованием этого управляющего символа удается уничтожить ключевое слово PROGRAM, которое должно появляться при загрузке Бейсик-блока. Достигается это с помощью полного "стирания" данного слова с помощью COMMA CONTROL.

В самом деле, если в 10 байтах хэдера, отведенных под заголовок, первой пойдет комбинация управляющих символов AT CONTROL выставляющая позицию печати в начале ключевого слова PROGRAM. После этого это слово сотрется, с использованием COMMA CONTROL. Далее позиция печати переносится в любое удобное для нас место экрана, например, в его середину опять-таки с использованием AT CONTROL и уже начиная с этого

места осуществляется печать названия программы. Схема распределения данных 10 байтов хэдера приведена на рисунке:

AT CNTR	row	col		AT CNTR	row	col			
1	2	3	4	5	6	7	8	9	10

Байты 1-3 занимает управляющий символ AT CONTROL со своими атрибутами. Байт 4 занимает символ COMMA CONTROL осуществляющий табуляцию в середину строки. Байт 5-7 занимает AT CONTROL с атрибутами. И всего лишь, 3 байта 8-10 остается под название программы.

* * *

Примечание "ИНФОРКОМА" даже и в этих 3 байтах можно разместить довольно длинные названия, используя токены ключевых слов.

Например:

RETURN TO RUN
LAND MOVE (L + AND + MOVE)
DRAW CAT !
STEP OVER BORDER
TANK (TAN + BACKSPACE + K)
RUN TO POINT
GO TO NEW POINT
TABOR (TAB * BACKSPACE + OR)
OUTRUN
OVERLIST
и т.д. и т.п.

Несмотря на простоту, подобное использование управляющих кодов достаточно неэкономично, но я надеюсь, что читатель в качестве эксперимента, укрепляющего свои познания в этой области, сумеет найти более рациональные решения, сохраняющие требуемый эффект.

2.3.5. Практическое использование управляющих кодов для защиты.

Мы рассмотрели теоретические основы применения управляющих кодов. Я постарался дать исчерпывающую информацию обо всех. О том, как это у меня получилось, судить Вам, читатель. В этом разделе перед нами стоит задача освоить предложенные теоретические идеи на практике. Если Вы хорошо поняли предыдущий материал, то и этот освоите без труда. Итак, все по порядку.

Управляющие коды "СПЕКТРУМа" можно получить несколькими способами. Первый - самый легкий, понятный и доступный - использование встроенной функции CHR.

Как Вам уже вероятно известно, все существующие на клавиатуре ключевые слова и символы, а также определенные потребителем графические символы совместно с управляющими кодами образуют полный набор символов "ZX SPECTRUM". Используя CHR и номер кода символа, можно получить строчное изображение каждого символа. Такое правило будет соблюдаться для всех символов, кроме управляющих кодов. Если вместе с CHR мы наберем управляющий код, то никакой новый символ на экране не появится, а будет выполнено действие, соответсвующее данному управляющему коду.

Рассмотрим применение CHR с управляющими кодами на базе оператора PRINT. Существует несколько вариантов использования CHR. После CHR и значения указанного кода может следовать точка с запятой, например:

```
PRINT  "A" ; CHR 6; "B"
```

Этот оператор выдает индикацию на экране:

```
A         B
```

т.к. используется управляющий код COMMA CONTROL. Использовать управляющие коды CHR можно и иначе - выстраивая из них составную строку. Так, оператор:

```
PRINT "A" + CHR 6 + "B"
```

даст тот же эффект, что и приведенный ранее пример.

Как мы уже знаем, коды от 16 до 23 управляют цветами и позицией печати.

Каждый из них может использоваться в составной строке. После CHR 16 (INK CONTROL) и CHR 17 (PAPER CONTROL) должны следовать CHR с указанным параметром цвета, а CHR 18... CHR 21 (FLASH, BRIGHT, INVERSE и OVER CONTROL) должны применяться вместе с CHR 0, CHR 1 или CHR 8.

Так, команда PRINT CHR 16 + CHR 2 + CHR 17 + CHR 6 + CHR 18 + CHR 1+ "SINCLAIR" изобразит на экране надпись "SINCLAIR" мерцающую красным и желтым цветами. Как и в случае, рассмотренном ранее, каждый знак плюс может быт заменен точкой с запятой.

После CHR 22 (AT CONTROL), как мы уже знаем, должны следовать два ключевых слова CHR с параметрами, указывающими номера строки и столбца позиции печати.

Так команда

```
PRINT CHR 22 + CHR 11 + CHR 16 + "W"
```

отпечатает в середине экрана символ "W".

После CHR 23 (TAB CONTROL) могут следовать 2 символа - это дает любопытный эффект: первое указывает позицию TAB, а второе обычно равно 0.

Так команда

```
PRINT CHR 23 + CHR 16+ CHR 0+"S"
```

отпечатает в середине экрана символ "S".

Как видите, такое применение данных кодов достаточно примитивно, а техника программирования, как Вы понимаете, не стоит на месте. Был создан более эффективный способ внедрения управляющих кодов, позволяющий обойтись без CHR.

Я не располагаю информацией о том, кто и когда впервые начал использовать управляющие коды и кто впервые использовал их для защиты. Но мне известно одно, а именно: ни одна современная программа к "ZX SPECTRUM" не обходится без применения управляющих кодов в своей защите. В связи с этим, возникают два вопроса:

1. Как научиться ставить защиту использующую управляющие коды, аналогичную применяемым в фирменных программах?

2. Как научиться быстро и просто снимать эту защиту с фирменных программ?

Рассмотрим теперь практическое применение управляющих кодов. Итак по-порядку:

BACKSPACE

Незаменим, когда вам необходимо скрыть какую-либо информацию Бейсика, например номер строки или некоторые другие атрибуты. За счет обратной табуляции Вы сможете также сокрыть от посторонних глаз особо ценную информацию.

Рассмотрим конкретный пример. Предположим у Вас имеется строка вида:

```
1 REM IT'S A FANTASTIC TO USE BACKSPACE
```

а мы бы желали, чтобы во время листинга на экране не было видно ни номера строки, ни Бейсик-оператора REM. Для этого вам необходимо изменить текст исходной строки, а именно: ввести между оператором REM и исходной надписью 6 символов (не имеет значение каких), чтобы потом заменить их на управляющий код BACKSPACE. Причем между REM и вставленными символами, а также между вставленными символами не должно быть пробелов.

Теперь нам необходимо символьную комбинацию, которую мы добавили между REM и текстом заменить управляющим кодом BACKSPACE. Для этого вычислим адрес, по которому расположен первый символ данной комбинации.

Для этого используем известный нам факт, что область Бейсика начинается с адреса 23755 (это условие верно только в том случае, когда к компьютеру не подключен INTERFACE 1 с микродрайвом и некоторая другая периферия). В случае сомнений найдите этот адрес для своей системы сами, используя системную переменную PROG (23635, 2 байта):

```
PRINT PEEK 23635 + 256*PEEK 23636
```

Количество символов, которое нам необходимо пропустить, равно пяти (4 символа это 2 байта номера строки и 2 байта длина строки + код оператора REM). Найдем адрес

ячейки, в которой хранится первый символ нашей комбинации:

23755+5=23760

Проверим это, подав команду:

```
PRINT CHR PEEK 23760
```

После этого заменим зарезервированную комбинацию управляющим кодом BACKSPACE. Подадим команды с клавиатуры:

```
FOR i=23760 TO 23765:
POKE i,8:
NEXT i
```

После чего содержимое нашей команды исчезнет с экрана, но оно останется в памяти и будет обрабатываться интерпретатором.

Для тех, кто желает оставить "надпись-памятку" для "хаккеров", рекомендую вернуться к примерам раздела 2.3.2. (самый первый шаг).

Практическое применение управляющих кодов INK CONTROL и PAPER CONTROL лучше всего описывать вместе, поскольку именно такое их использование приносит наилучший эффект. Итак, по-порядку.

В защите применение INK CNTR и PAPER CNTR наиболее эффективно, когда нам необходимо скрыть текст листинга. Достигается это с помощью подачи команд, задающих одинаковый цвет INK и PAPER. Рассмотрим более подробно, как это сделать.

Предположим, Вы желаете сделать так, чтобы после подачи команды LIST экран оставался пустым. Для этого нам необходимо первой строкой программы поставить такую строку, которая бы "уничтожала" номер строки и изменяла цвета INK и PAPER управляющими кодами INK CONTROL и PAPER CONTROL. Чтобы стереть номер строки на экране нам понадобится использовать управляющий код BACKSPACE.

Итак, для начала зарезервируем место, используя оператор REM, чтобы потом спокойно заполнять его управляющими кодами. Наберем

```
1 REM HHHHHHHHH
       9символов
```

Нам необходимо зарезервировать место именно для 9 символов т.к. первые 5 заменит BACKSPACE, а остальные 2+2 мы используем для INK и PAPER CONTROL с соответствующими параметрами таким образом, чтобы сделать одинаковыми цвета символов и фона.

Для ввода первых пяти символов BACKSPACE подадим команду с клавиатуры:

```
FOR i=23760 TO 23764:
POKE i,8:
NEXT i.
```

Для задания черного цвета INK подадим команды

```
POKE 23765,16: POKE 23765,0
```

Для задания черного цвета PAPER подадим команды

```
POKE 23767,17: POKE 23768,0
```

Теперь следующие строки вводимые после этого операторы не должны быть видны. Введем первый оператор, создав новую Бейсик-строку и убедимся, что это соответствует действительности.

Теперь рассмотрим одно из наиболее любопытных применений управляющего кода AT CONTROL. Многим из Вас, наверняка доводилось встречать программы, при загрузке которых на экране появлялось лишь одно название программы (имеется ввиду, что отсутствовало слово PROGRAM), либо название программы появлялось в необычном месте экрана, например в середине. Достигалось это использованием кода AT CONTROL, а в первом случае с применением COMMA CONTROL. Итак, по-порядку.

Для того, чтобы сделать такой заголовок, нам необходимо записать на ленту хэдер, содержащий управляющие коды.

Поскольку, сначала необходимо создать такой хэдер, то возникает, вопрос: "Как это сделать?", - ввиду того, что набор управляющих кодов с клавиатуры проблематичен.

Я предлагаю сделать следующее: после того, как Вы записали свою программу на

ленту с любым названием, содержащим 10 символов, загрузить Ваш хэдер в копировщик COPY - COPY, найти там адрес ячеек, содержащих название, используя команду LIST и поменять их командой POKE, следуя нижеизложенным рекомендациям.

Для тех, кто не имеет копировщика COPY - COPY, возможен другой вариант. Одной из строк своей программы Вы делаете строку

```
nn SAVE "имя программы" LINE m
```

После этого Вам необходимо получить дампинг памяти, используя одну из предложенных в этой главе программ. Запомнив адреса ячеек памяти, в которых находятся байты с названием Вашей программы, необходимо изменить их, следуя нижеизложенным рекомендациям, после чего обратиться к этой строке командой GO TO nn и записать файл в магнитофон. Неудобство второго метода заключается в том, что созданная Вами строка nn SAVE "имя программы" LINE m, должна будет остаться в программе, т.к. при записи хэдера в общую длину программы вошла также и длина этой строки.

Итак, какие надо делать изменения, чтобы название печаталось в произвольном месте экрана? Необходимо лишь задать данную позицию, используя управляющий код AT CONTROL. Из отводящихся под название 10 байтов, 3 будет занимать AT CONTROL со своими параметрами. Ввиду этого под само название остается лишь 7 байтов.

Для того, чтобы распечатать название NEITRON в позициях AT 10,12, Вам необходимо, чтобы в 10 байтах хэдера, отведенных под название, содержалась следующая комбинация:

22	10	12	78	69	73	84	82	79	78
AT CNTR	r	c	N	E	I	T	R	O	N
1	2	3	4	5	6	7	8	9	10

Если же мы хотим создать систему команд, уничтожающую слово PROGRAM, то необходимо действовать в следующей последовательности:

1) Установить курсор в позицию AT 1,0 используя код AT CONTROL, чтобы следующей командой стереть это слово.

2) Стереть слово PROGRAM, используя управляющий код COMMA CONTROL.

3) Установить курсор в то место экрана, с которого мы желали бы распечатать текст названия программы с помощью AT CONTROL.

4) Распечатать название программы.

Таким образом, как видим, в предложенном варианте под текст названия программы остается всего 3 знакоместа. Для тех, кому этого окажется мало, можно не делать пункт 3 данного плана, что позволяет сэкономить еще 3 байта. Итак, чтобы распечатать название программы MVS в позиции экрана 8, 8, уничтожив перед этим слово PROGRAM необходимо, чтобы 10 байтов заголовка имели вид:

22	1	0	6	22	8	8	77	86	83
AT CNTR	r	c	COM CNTR	AT CNTR	r	c	M	V	S
1	2	3	4	5	6	7	8	9	10

Возможно, что экспериментируя над управляющими кодами, читателю удастся обнаружить нечто новое в этой интересной области.

Теперь некоторые советы. Выше было описано использование управляющих кодов в операторе PRINT через CHR. Я полагаю, что Вам будет удобней экспериментировать именно используя непосредственно управляющие коды.

И последнее. Запомните, пожалуйста, что байт, стоящий после управлявшего кода, содержит именно значение данной цифры, а не её ASCII код. Это правило касается абсолютно всех управляющих кодов.

ПРОГРАММА ДЛЯ СНЯТИЯ ЗАЩИТ

Предназначена для программ, которые используют управляющие коды INK CONTROL и PAPER CONTROL.

```
9990 REM Программист МИХАЙЛЕНКО ВАДИМ МРТИ 010207,1991
9991 PAPER 7: INK 0: BORDER 7: CLS
```

```
9992 FOR I=23758 TO 65000
9993 IF PEEK I =13 THEN IF PEEK (I+1)=39 AND PEEK (I+2)=6 THEN STOP
9994 IF PEEK I=13 THEN LET I=I+4
9995 IF PEEK I=16 THEN POKE (I+1),0: LET I=I+2
9996 IF PEEK I=17 THEN POKE (I+1),7: LET I=I+2
9997 NEXT I
```

Краткое описание.

Задав необходимые значения цветов INK, PAPER и BORDER, программа в цикле начинает анализировать содержимое каждой ячейки памяти и, если встречается код INK CONTROL то цвет символов принудительно задаётся чёрным, а для PAPER CONTROL - белым.

Строка 9993 следит за тем, чтобы как только программа дойдёт до самой себя (до строки 9906), произошла остановка, поэтому наличие этой строки - обязательно.

Глава 3. Методы защиты от MERGE

Среди приемов, наиболее часто применяемых для взлома программ, одним из распространенных является использование MERGE вместо команды LOAD"". Это позволяет загрузить программу в память компьютера без ее автостарта.

Таким образом, в случае наличия в программе POKES, которые защищают от BREAK, либо делают листинг программы нечитаемым, их удается разблокировать.

Автостарт должен был бы запустить программу, что позволило бы организовать защиту посредством POKES, но MERGE загружает программу без автостарта.

Вот почему одним из первоочерёдных условий защиты является создание системы команд, способных защитить Вашу программу от использования MERGE"".

Для того, чтобы уяснить себе, как действуют данные методы защиты, необходимо проанализировать выполнение функции MERGE. Итак, рассмотрим, как работает этот раздел интерпретатора Бейсика. Как Вам уже вероятно известно, встроенные в ПЗУ программы обработки LOAD, SAVE, VERIFY и MERGE работают вместе, используя многие общие процедуры. Команда MERGE очень близка по своей сути к команде LOAD, естественно с некоторыми специфическими отличиями. В частности, MERGE загружает файл Бейсика в специальный буфер. После загрузки интерпретатор начинает анализировать последовательно каждую строку загруженной программы. Если все проанализированное с точки зрения интерпретатора верно, то происходит непосредственное выполнение команды MERGE, заключающееся в соединении двух программ в одну, причем новая программа затирает строки с теми же номерами, а также переменные с теми же именами.

Но это происходит лишь в том случае, если до загрузки командой MERGE в памяти уже находилась какая-либо программа. Если же мы загружаем посредством MERGE новую программу в очищенную память, то, естественно, никакого слияния не происходит, а программа просто загружается в память, анализируется интерпретатором и после этого компьютер останавливает свою работу, позволяя нам просмотреть листинг данной программы. Так использование MERGE вместо LOAD позволяет загрузить программу без автостарта.

Быть может, теперь у нетерпеливого читателя возникнет желание просмотреть все фирменные программы, используя эту команду. Что ж, попробуйте, но необходимо заметить, что в большинстве из них стоит защита от MERGE. Необходимость такой защиты очевидна из-за описанных выше возможностей для хаккеров просматривать программы. Поэтому профессионалы достаточно широко используют данный метод защиты.

Рассмотрим теоретические аспекты работы защит от MERGE.

Как вам уже известно, работа оператора MERGE заключается в том, что он загружает программу в специальный буфер и там её анализирует. Если анализ проходит успешно, то никаких сбоев не будет, но если он проходит не "очень гладко", то компьютер просто-напросто зависает. То есть, как видим, задача программиста, разрабатывающего защиту от MERGE, состоит в том, чтобы его программа имела "встроенный дефект", который бы после

загрузки с помощью MERGE проявлял себя. В то же время необходимо отметить, что этот "дефект" не должен никак проявляться после загрузки программы командой LOAD"".

Одним из методов, позволяющих осуществить подобную защиту, является задание фальшивого номера строки параллельно с фальшивой длиной строки. Фальшивый номер и длина строки сразу же проявятся, как только интерпретатор начнет анализировать программу во время выполнения команды MERGE.

Но существует один нюанс, который тоже необходимо учитывать. Интерпретатор анализирует и ту строку программы, которую он выполняет во время работы программы. А это значит, что если после загрузки программы интерпретатору Бейсика попадается созданная нами строка, то работа компьютера будет прервана по ошибке:

"Ошибка в Бейсике (NONSENSE IN BASIC)"

Следовательно, нам необходимо создать данную строку таким образом, чтобы она никогда не обрабатывалась интерпретатором во время работы программы. Одним из методов, позволяющим сделать это, является размещение данной строки в самом конце программы.

Рассмотрим, как это осуществить на практике.

Предположим, Вами создана программа, причем одним из необходимых условий является то, чтобы номера последних шагов программы не превышали 9980, поскольку в этой области памяти мы разместим специальную программу, которая создаст вам требуемую строку защиты от MERGE.

```
9985 FOR I=23758 TO 65000
9986 IF PEEK I=13 THEN IF PEEK (I+1)=39 AND PEEK (I+2)=6 THEN POKE (I+1),255: POKE(I+2),255:
     POKE(I+3),255: POKE(I+4),255: PRINT "THE END": STOP
9987 NEXT I
9990 REM защита от MERGE
9994 REM ПРОГРАММИСТ МИХАИЛЕНКО ВАДИМ, МЕНСК, МРТИ, ГР 010207.
```

Данная программа осуществляет принудительное изменение одного из номеров содержащихся в ней же строк. Номер этой строки - 9990, так что наличие ее в программе - обязательно. Если читатель хочет оставить памятку - сообщение для взломщиков, то он может написать в этой строке после REM произвольный текст, достаточно убедительно показывающий непосвященному, что программа защищена именно Вами. Подобные комментарии постоянно оставляет после себя один из наиболее известных в нашей стране "хаккеров" BILL GILBERT, причем он использует именно этот метод защиты от MERGE"".

Программа действует следующим образом. В цикле анализируется содержимое текущей ячейки памяти. Если оно равно 13, то, следовательно, это код ENTER, а это, в свою очередь, не что иное, как окончание данной строки. А раз данная строка окончена, то, зная структуру Бейсик-программы, мы можем утверждать, что после этого кода следуют 4 байта, ответственные за номер следующей строки и за ее длину. Раз так, тогда анализируем номер строки. Если его кодовое представление равно 9990, - именно этот номер нам необходим, то тогда принудительно засылаем в ячейки Бейсика, ответственные за номер, значение 255, чтобы умышленно изменить содержащиеся там правильные значения. После того, как эта операция закончится, на экране печатается "THE END" и программа останавливается.

После того, как Вы создали "защитную" строку Вы уже не увидите ее на экране. Но она будет последней строкой и Вам необходимо строго соблюдать условие, чтобы интерпретатор Бейсика данную строку не анализировал.

После того, как предложенная программа выполнила возложенную на нее миссию - создала строку защиты от MERGE, её необходимо уничтожить, подав команды:

```
9984 ENTER
9985 ENTER
9986 ENTER  и т.д.
```

Заканчивая рассмотрение системы защиты от MERGE, хотелось бы подчеркнуть тот факт, что здесь был описан лишь один из многочисленных приемов, применяющихся для создания защитной строки на Бейсике, препятствующих применению MERGE. Но то, что эта программа производит автоматически, можно осуществлять и "вручную", если использовать специальную программу для получения "дампа" памяти, описанную в предыдущей Главе.

Существуют методы защиты, достаточно кардинально отличающиеся от описанного выше.

Мы уже говорили о том, что в БЕЙСИК-строке можно разместить программу в машинных кодах. Это используется с помощью применения оператора Бейсика REM и описано в ГЛАВЕ I. Там же указано, что для нормальной работы интерпретатора необходимо, чтобы программа в машинных кодах не содержала кода перевода строки - 13. Это объясняется тем, что интерпретатор, анализируя данную строку Бейсика, определяет ее окончание именно по коду 13 и, если он его находит, то следующие за ним 4 байта он рассматривает, как номер и длину следующей, БЕЙСИК-строки. А эти байты, естественно, не могут правильно характеризовать БЕЙСИК-строку и, следовательно, должен произойти сбой в работе интерпретатора. Это очень напоминает ситуацию, когда мы пытались сделать фальшивыми номер и длину первой идущей в программе строки. То же самое происходит и в нашей программе, если мы используем в ней машинные коды, в которых есть код 13.

В этом есть некое рациональное зерно. Преимущества налицо, если у Вас есть программа, в которой на Бейсике сформирована строка машинных кодов, где присутствует код 13 (если он отсутствует, то его можно внедрить в программу, используя специальную директиву АССЕМБЛЕРа DEFB). В этом случае Вам уже не понадобится формировать специальную строку, ответственную за защиту от MERGE, что способствует экономии оперативной памяти, а это очень немаловажно для Бейсика, особенно если это загрузчик.

В то же время, следует учитывать, что этот метод может оказаться недействительным, если программа в кодах будет стоять в первой строке БЕЙСИКа. В этом и заключена одна из сложностей применения данного метода, поскольку интерпретатор начинает анализировать программу в машинных кодах, как несколько строк БЕЙСИК-программы, разделенных кодом 13. В этом случае он может остановить свою работу и выдать код сообщения об ошибке:

```
NONSENSE IN BASIC
```

Избавиться от проблемы можно, если сделать, чтобы машинные коды шли последней строкой программы. Осуществляется это достаточно просто: необходимо при создании своей программы лишь соблюсти определенную последовательность операций, которая и приведет к желаемому результату:

- Сначала формируем строку с машинными кодами, но теперь при формировании даем ей такой номер, чтобы все строки нашей исходной программы размещались до нее.

Причем необходимо, чтобы до начала следующего пункта нашего плана эти строки были сформированы. Здесь необходимо отметить тот факт, что даже в команде запуска программы в машинных кодах (RANDOMIZE USR NN) должны присутствовать значения цифр и именно такие, чтобы они приблизительно указывали на место расположения Вашей программы в кодах. Это необходимо для того, чтобы ваша программа имела точно фиксированное место расположения в оперативной памяти компьютера. После того, как мы в следующем пункте найдем точку старта подпрограммы в кодах, нам останется лишь заменить адрес в команде RANDOMIZE USR на правильный, но количество символов, соответствующих номеру старта не должно поменяться, т.к. уменьшение или увеличение числа цифр в номере приведет к смещению на один байт нашей программы в кодах. В общем случае после команды RANDOMIZE USR должно следовать пятизначное число, указывающее на какой-либо адрес, а после обнаружения правильного значения, это число соответственно откорректируется.

Для того, чтобы обнаружить истинное место в оперативной памяти нашей программы, можно использовать несколько методов:

1 - В частности, поручить это делать компьютеру, составив соответствующую программу.

2 - Сделать просмотр памяти "вручную".

В любом из этих случаев нам понадобится вводить дополнительные строки в Бейсик, а делать это необходимо таким образом, чтобы они размещались после строки, в которой содержится подпрограмма в кодах (используя такой прием, мы не изменим местоположение этой строки с подпрограммой в оперативной памяти).

В первом из рассмотренных случаев нам понадобится составить программу, чтобы

она самостоятельно в цикле анализировала бы область Бейсика с тем, чтобы обнаружить там номер строки в кодах. При обнаружение этой строки она должна распечатать номер с учетом 4-х байтов, отводящихся под номер и длину, и с учетом пятого байта, отведенного под REM. Выше было приведено достаточное количество подобных программ анализаторов, так что я надеюсь, что читатель сам справится с этой задачей.

Второй случай аналогичен первому с той разницей, что вам придется использовать программу получения дампинга, которая также была приведена в предыдущей главе.

Естественно, что после того, как эта программа выполнит свою задачу, ее строки необходимо будет уничтожить.

(Продолжение следует).

40 ЛУЧШИХ ПРОЦЕДУР

Мы продолжаем печать книгу Дж. Хардмана и Э. Хьюзона. Сегодня вашему вниманию предлагается подробный разбор еще 15 полезных процедур для самообразования.

Начало см. "ZX-РЕВЮ" N1-2, стр. 17-28.

5.8 Сдвиг вниз на один символ.

Длина: 73

Количество переменных: 0

Контрольная сумма: 7987

Назначение: Эта программа сдвигает содержимое дисплейного файла вниз на 8 пикселей.

Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Нет

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22527        33    255   87
            LD DE,22495        17    223   87
SAVE        PUSH HL            229
            PUSH DE            213
            LD C,23            14    23
NEXT_L      LD B,32            6     32
COPY_B      LD A,(DE)          26
            LD (HL),A          119
            LD A,C             121
            AND 7              230   7
            CP 1               254   1
            JR NZ,NEXT_B       32    2
            SUB A              151
            LD (DE),A          18
NEXT_B      DEC HL             43
            DEC DE             27
            DJNZ COPY_B        16    241
            DEC C              13
            JR Z,REST          40    21
            LD A,C             121
            AND 7              230   7
            CP 0               254   0
            JR Z,N_BLOCK       40    24
            CP 7               254   7
            JR NZ,NEXT_L       32    225
            PUSH DE            213
            LD DE,1792         17    0     7
            AND A              167
            SBC HL,DE          237   82
            POP DE             209
            JR NEXT_L          24    215
REST        POP DE             209
            POP HL             225
            DEC D              21
            DEC H              37
            LD A,H             124
            CP 79              254   79
```

```
            RET Z               200
            JR SAVE             24     201
N_BLOCK     PUSH HL             229
            LD HL,1792          33     0      7
            EX DE,HL            235
            AND A               167
            SBC HL,DE           237    82
            EX DE,HL            235
            POP HL              225
            JR NEXT_L           24     193
```

Как она работает:

В пару регистров HL загружается адрес последнего байта дисплейного файла, а в DE загружается адрес байта, соответствующего изображению, отстоящему на восемь линий вверх. HL и DE сохраняются на стеке. В C-регистр загружается число, на 1 меньшее, чем число строк на экране. Затем в B-регистр загружается количество байтов на одной линии дисплея - он используется, как счетчик. В аккумулятор загружается байт с адресом DE и это значение пересылается в ячейку по адресу HL. В аккумулятор загружается (для проверки) содержимое регистра C и, если оно равно 1, 9 или 17, то в ячейку по адресу DE помещается 0. HL и DE уменьшаются на единицу, указывая на следующий байт дисплея. Счетчик байтов в регистре B уменьшается и, если и он не равен 0, происходит переход к COPY_B.

Далее уменьшается счетчик строк в регистре C. Если он равен 0,происходит переход к процедуре 'REST'. Если C содержит 8 или 16, то происходит переход к процедуре 'N_BLOCK'. Если C не содержит 7 или 15, подпрограмма переходит к 'NEXT_L'. Затем из HL вычитается 1792 - теперь HL указывает на следующую треть экрана и подпрограмма переходит к 'NEXT_L'.

В процедуре REST значения DE и HL берутся из стека и из них вычитается число 256. В итоге DE и HL указывают на строку, позиция которой выше, чем та, что была в предыдущем цикле. Если HL содержит 20479, подпрограмма возвращается в BASIC, иначе происходит переход к процедуре SAVE.

В процедуре N_BLOCK, 1792 вычитается из DE - т.е. после этого DE указывает на следующий блок экрана. Подпрограмма затем переходит к NEXT_L.

5.9 Сдвиг влево на один пиксел.

Длина: 17

Количество переменных: 0

Контрольная сумма: 1828

Назначение: Сдвиг содержимого дисплейного файла на один пиксел влево.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарии: Эта программа осуществляет более плавное перемещение, чем сдвиг влево на один символ, но требуется вызывать ее восемь раз, чтобы переместить дисплей на один полный символ.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,22527         33     255    87
            LD C,192            14     192
NEXT_L      LD B,32             6      32
            OR A                183
NEXT_B      RL (HL)             203    22
            DEC HL              43
            DJNZ NEXT_B         16     251
            DEC C               13
            JR NZ,NEXT_L        32     245
```

```
        RET                 201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта дисплейного файла, а в регистр C загружается количество линий в дисплейном файле (он используется, как счетчик линии). В регистр "B" загружается количество байтов на одной линии (он используется, как счетчик байтов). Флаг переноса устанавливается в 0.

Байт, адресованный HL, сдвигается на один бит влево, бит переноса копируется в крайний правый бит, а крайний левый бит копируется во флаг переноса. Пара регистров HL уменьшается для указания на следующий байт, и счетчик (B-регистр) уменьшается. Если и он не равен 0, подпрограмма осуществляет переход к NEXT_B, иначе уменьшается количество обрабатываемых линий, и, если оно не равно 0, подпрограмма возвращается к процедуре NEXT_L.

По окончании работы программа возвращается в BASIC.

5.10 Сдвиг вправо на один пиксел.

Длина: 17

Количество переменных: 0

Контрольная сумма: 1550

Назначение: Сдвиг содержимого дисплейного файла на один пиксел вправо.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Эта программа осуществляет более плавное перемещение, чем сдвиг вправо на один символ, но требуется восемь раз вызывать эту программу, чтобы переместить дисплей на один полный символ.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,16384         33     0      64
            LD C,192            14     192
NEXT_L      LD B,32             6      32
            OR A                183
NEXT_B      RR (HL)             203    30
            INC HL              35
            DJNZ NEXT_B         16     251
            DEC C               13
            JR NZ,NEXT_L        32     245
            RET                 201
```

Как она работает:

В пару регистров HL загружается адрес дисплейного файла, а в C-регистр загружается количество линий на экране (он используется, как счетчик линий). В регистр "B" загружается количество байтов на одной линии (он используется, как счетчик байтов). Флаг переноса устанавливается, в 0. Байт по адресу HL сдвигается на один бит вправо, бит переноса копируется в крайний левый бит, а крайний правый бит копируется во флаг переноса. Пара регистров HL увеличивается, указывая на следующий байт, и счетчик (B-регистр) уменьшается.

Если он не равен 0, подпрограмма осуществляет переход к NEXT_B, иначе уменьшается количество обрабатываемых линий, и, если оно не равно 0, подпрограмма возвращается к NEXT_L.

Закончив работу, процедура возвращается в BASIC.

5.11 Сдвиг вверх на один пиксел.

Длина: 91

Количество переменных: 0
Контрольная сумма: 9228
Назначение: Сдвиг содержимого дисплейного файла вверх на один пиксел.
Вызов программы: RANDOMIZE USR адрес
Контроль ошибок: Нет
Комментарий: Нет

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

           LD HL,16384        33    0     64
           LD DE,16640        17    0     65
           LD C,192           14    192
NEXT_L     LD B,32            6     32
COPY_B     LD A,(DE)          26
           LD (HL),A          119
           LD A,C             121
           CP 2               254   2
           JR NZ,NEXT_B       32    2
           SUB A              151
           LD (DE),A          18
NEXT_B     INC DE             19
           INC HL             35
           DJNZ COPY_B        16    243
           PUSH DE            213
           LD DE,224          17    224   0
           ADD HL,DE          25
           EX (SP),HL         227
           ADD HL,DE          25
           EX DE,HL           235
           POP HL             225
           DEC C              13
           LD A,C             121
           AND 7              230   7
           CP 0               254   0
           JR NZ,SUBTR        32    10
           PUSH DE            213
           LD DE,2016         17    224   7
           AND A              167
           SBC HL,DE          237   82
           POP DE             209
           JR N_BLOCK         24    14
SUBTR      CP 1               254   1
           JR NZ,N_BLOCK      32    10
           PUSH HL            229
           EX DE,HL           235
           LD DE,2016         17    224   7
           AND A              167
           SBC HL,DE          237   82
           EX DE,HL           235
           POP HL             225
N_BLOCK    LD A,C             121
           AND 63             230   63
           CP 0               254   0
           JR NZ,ADD          32    6
           LD A,7             62    7
           ADD A,H            132
           LD H,A             103
           JR NEXT_L          24    187
ADD        CP 1               254   1
           JR NZ,NEXT_L       32    183
           LD A,7             62    7
```

```
        ADD A,7         130
        LD D,A          87
        LD A,C          121
        CP 1            254   1
        JR NZ,NEXT_L    32    174
        RET             201
```

Как она работает:

В пару регистров HL загружается адрес дисплейного файла, а в DE - адрес первого байта второй строки дисплея. В регистр "C" загружается число линий на экране. В регистр "B" загружается количество байтов на одной строке он используется как счетчик.

В аккумулятор загружается байт с адресом в DE и это значение передается в ячейку по адресу HL. В аккумулятор заносится содержимое регистра "C". Если оно равно 2, то DE указывает на нижнюю строку экрана и по этому адресу записывается 0. HL и DE увеличиваются, чтобы указать на следующий байт. Счетчик в регистре "B" уменьшается и, если он не равен 0, происходит переход к COPY_B.

224 добавляется к регистровым парам HL и DE, чтобы они указывали на следующую строку экрана. Затем уменьшается регистр C, счётчик линий. Если содержимое регистра C не кратно 8, происходит переход к SUBTR иначе 2016 вычитается из HL и происходит переход к N_BLOCK. Теперь HL указывает на следующие 8 линий.

В процедуре SUBTR, если значение (C-1) не кратно 8, происходит переход к N_BLOCK, иначе 2016 вычитается из DE, т.е. теперь DE указывает на следующие 8 линий. В процедуре N_BLOCK, если значение C-регистра кратно 64, 1792 добавляется к HL и делается переход NEXT_LINE - теперь HL указывает на следующий блок из 64 линий. В процедуре ADD, если значение (C-1) не кратно 64,1792 добавляется к DE, чтобы пара DE указывала на следующий блок 64 линий. Если C-регистр не содержит 1, то программа возвращается к N_LINE иначе - возврат в BASIC.

5.12 Сдвиг вниз на один пиксел.

Длина: 90

Количество переменных: 0

Контрольная сумма: 9862

Назначение: Сдвиг содержимого дисплейного файла вниз на один пиксел.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Нет

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА   АССЕМБЛЕР       ЧИСЛА ДЛЯ ВВОДА

        LD HL,22527     33    255   87
        LD DE,22271     17    255   86
        LD C,192        14    192
NEXT_L  LD B,32         6     32
COPY_B  LD A,(DE)       26
        LD (HL),A       119
        LD A,C          121
        CP 2            254   2
        JR NZ,NEXT_B    32    2
        SUB A           151
        LD (DE),A       18
NEXT_B  DEC DE          27
        DEC HL          43
        DJNZ COPY_B     16    243
        PUSH DE         213
        LD DE,224       17    224   0
        AND A           167
```

```
         SBC HL,DE          237   82
         EX (SP),HL         227
         AND A              1     67
         SBC HL,DE          237   82
         EX DE,HL           235
         POP HL             225
         DEC C              13
         LD A,C             121
         AND 7              230   7
         CP 0               254   0
         JR NZ,ADD          32    8
         PUSH DE            213
         LD DE,2016         17    224   7
         POP DE             209
         JR N_BLOCK         24    11
         ADD CP 1           254   4
         JR NZ,N_BLOCK      32    7
         PUSH HL            229
         LD HL,2016         33    224   7
         ADD HL,DE          25
         EX DE,HL           235
         POP HL             225
N_BLOCK  LD A,C             121
         CP 0               254   0
         JR NZ,SUBTR        32    6
         LD A,H             124
         SUB 7              214   7
         LD H,A             103
         JR NEXT_L          24    188
SUBTR    CP 1               254   1
         JR NZ,NEXT_L       32    184
         LD A,D             122
         SUB 7              214   7
         LD D,A             87
         LD A,C             121
         CP 1               254   1
         JR NZ,NEXT_L       32    175
         RET                201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта дисплейного файла, а в пару регистров DE загружается адрес байта линии, которая находится непосредственно над этим байтом. В регистр "C" загружается число линий экрана. В регистр "B" загружается количество байтов на одной строке дисплея - он используется, как счетчик.

В аккумулятор загружается байт из ячейки с адресом в DE, и это значение загружается в ячейку по адресу в HL. В аккумулятор загружается содержимое регистра "C" - если оно равно 2, то DE указывает на верхнюю строку экрана, в адрес которой заносится 0. HL и DE уменьшаются, чтобы указать на следующие байты. Счетчик (B-регистр) уменьшается и, если он не равен 0, происходит переход к COPY_B.

224 вычитается из регистровых пар HL и DE - теперь они указывают на следующую строку экрана. Регистр C (счетчик строк) уменьшается. Если содержимое регистра C кратно 8, т.е. выполнено 8 циклов для одной строки, происходит переход к процедуре ADD, иначе 2016 добавляется к HL и происходит переход к N_BLOCK - HL теперь указывает на следующие 8 линий.

В процедуре ADD, если значение (C-1) не делится без остатка на 8, происходит переход к N_BLOCK, в противном случае 2016 добавляется к DE, чтобы пара регистров DE указывала на следующие 8 линий. В процедуре N_BLOCK, если значение C - регистра делится без остатка на 64, из HL вычитается 1792 - HL теперь указывает на следующий блок из 64 линий, и происходит переход к NEXT_L. В процедуре SUBTR, если значение (C-1) не кратно 64, из DE вычитается 1792 - в итоге DE указывает на следующий блок из 64 линий.

Если C-регистр не содержит 1, то подпрограмма возвращается к NEXT_LINE, иначе - в BASIC.

6.ДИСПЛЕЙНЫЕ ПРОГРАММЫ

6.1 Слияние картинок

Длина: 21

Количество переменных: 1

Контрольная сумма: 1709

Назначение: Эта программа объединяет картинку, хранящуюся в ОЗУ, с текущим экраном дисплея. Атрибуты не изменяются.

Переменные:

Имя - screen_store

Длина - 2

Адрес - 23296

Комментарий: содержит адрес картинки в ОЗУ.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарии: Для объединения картинок должна быть использована приведенная на листинге программа, однако интересные результаты могут быть также получены заменой OR (HL) на XOR (HL) или AND (HL).

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            LD HL,16384        33    0     64
            LD DE,(23296)      237   91    0     91
            LD BC,6144         1     0     24
NEXT_B      LD A,(DE)          26
            OR (HL)            182
            LD (HL),A          119
            INC HL             35
            INC DE             19
            DEC BC             11
            LD A,B             120
            OR C               177
            JR NZ,NEXT_B       32    246
            RET                201
```

Как она работает.

В пару регистров HL загружается начальный адрес дисплейного файла, а в пару регистров DE - его длина. Регистровая пара BC используется в качестве счетчика.

В аккумулятор загружается байт с адресом в DE и выполняется логическое 'ИЛИ' ('OR') этого значения с байтом дисплейного файла. Результат затем помещается в область дисплея.

HL и DE перемещаются на следующую позицию, счетчик уменьшается. Если счетчик не равен 0, то подпрограмма возвращается назад для повторения процесса со следующим байтом.

6.2. Инвертирование экрана.

Длина: 18

Количество переменных: 0

Контрольная сумма: 1613

Назначение: Инвертирует все в дисплейном файле: если пиксел включен - он сбрасывается (OFF), если пиксел выключен, он устанавливается.

Вызов программы:
```
RANDOMIZE USR адрес
```
Контроль ошибок: Нет

Комментарий: Эта программа может быть использована для получения эффекта вспышки. Этот эффект усиливается, если делается вызов несколько раз и добавляется звук.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА
            LD HL,16384         33     0      64
            LD BC,6144          1      0      24
            LD D,255            22     255
NEXT_B      LD A,D              122
            SUB (HL)            150
            LD (HL),A           119
            INC HL              35
            DEC BC              11
            LD A,B              120
            OR C                177
            JR NZ,NEXT_B        32     247
            RET                 201
```

Как она работает:

В пару регистров HL загружается адрес дисплейного файла, а в BC загружается его длина. В D-регистр помещается значение 255. Всякий раз, когда подпрограмма возвращается к NEXT_B, в аккумулятор загружается значение из D регистра. Этот метод предпочтительнее, чем инструкция LD A,255 т.к. LD A,D выполняется приблизительно в 2 раза быстрее, чем инструкция LD A,255. Значение байта, хранящегося в ячейке по адресу, указанному в HL, вычитается из аккумулятора, а результат загружается в тот же самый байт. Таким образом, делается инвертирование.

HL увеличивается, указывая на следующий байт, а счетчик BC, уменьшается. Если счетчик не равен 0, программа возвращается к NEXT_B. Если счетчик равен 0, программа возвращается в BASIC.

6.3 Инвертирование символа вертикально.

Длина: 20

Количество переменных: 1

Контрольная сумма: 1757

Назначение: Эта программа инвертирует символ вертикально. Например, стрелка, направленная вверх, должна стать стрелкой, направленной вниз, и наоборот.

Переменные:

Имя - chr. start

Длина - 2

Адрес - 23296

Комментарий: адрес символа в ОЗУ (ПЗУ).

Вызов программы:
```
RANDOMIZE USR адрес
```
Контроль ошибок: Нет

Комментарий: Эта программа полезна в играх, т.к. можно изменять отдельные символы, не затрагивая при этом соседние области изображения.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,(23296)       42     0      91
            LD D,H              84
            LD E,L              93
```

```
          LD B,8          68
NEXT_B    LD A,(HL)       126
          INC HL          35
          PUSH AF         245
          DJNZ NEXT_B     16    251
          LD B,8          68
REPL      POP AF          241
          LD (DE),A       18
          INC DE          19
          DJNZ REPL       16    251
          RET             201
```

Как она работает:

В пару регистров HL загружается адрес данных символа. Этот же адрес затем копируется в DE. В регистр B загружается значение 8 для использования регистра в качестве счетчика. Для каждого байта в аккумулятор загружается имеющееся в настоящий момент значение. HL увеличивается, указывая на следующий байт, а содержимое аккумулятора помещается на стек. Счетчик уменьшается, и, если он не равен 0, подпрограмма возвращается, чтобы повторить процесс для следующего байта. В регистр "B" повторно загружается значение 8, чтобы снова использовать его в качестве счетчика. Изображение символа сохранено на стеке.

Процедура REPL возвращает данные со стека на то же знакоместо, но уже в обратном порядке.

Байт за байтом берутся со стека и через аккумулятор помещаются по адресу, содержащемуся в DE. DE увеличивается, чтобы указать на следующий байт, а счетчик уменьшается. Если он не равен 0, программа возвращается к REPL. В противном случае она возвращается в BASIC.

6.4 Инвертирование символа горизонтально.

Длина: 19

Количество переменных: 1

Контрольная сумма: 1621

Назначение: Эта программа инвертирует символ горизонтально - например, стрелка, направленная влево, становится стрелкой, направленной вправо.

Переменные:

Имя - chr_start

Длина - 2

Адрес - 23296

Комментарий - адрес данных символа.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Нет

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА     АССЕМБЛЕР       ЧИСЛА ДЛЯ ВВОДА

          LD HL,(23296)   42    0     91
          LD A,8          62    8
NEXT_B    LD B,8          6     8
NEXT_P    RR (HL)         203   30
          RL C            203   17
          DJNZ NEXT_P     16    250
          LD (HL),C       113
          INC HL          35
          DEC A           61
          JR NZ,NEXT_B    32 243
```

```
        RET                 201
```

Как она работает: В пару регистров HL загружается адрес данных символа, а в аккумулятор загружается количество байтов, которые должны быть инвертированы. В регистр B загружается число битов в каждом байте он используется, как счетчик.

Байт с адресом в HL сдвигается вправо таким образом, что крайний правый бит копируется во флаг переноса. C-регистр сдвигается влево так, что флаг переноса копируется в крайний правый бит. Счетчик (B-регистр) уменьшается. Если счетчик не равен 0, происходит переход к NEXT_P для работы со следующим пикселом.

Инвертированный байт, который находится в регистре C, помещается в ячейку, из которой он был взят. HL увеличивается, указывая на следующий байт, а аккумулятор уменьшается. Если аккумулятор не равен 0, происходит переход к NEXT_BYTE, в противном случае - возврат в BASIC.

6.5 Вращение символа по часовой стрелке.

Длина: 42

Количество переменных: 1

Контрольная сумма: 3876

Назначение: Эта программа поворачивает символ на 90 градусов по часовой стрелке, например, стрелка, направленная вверх становятся направленной вправо.

Переменные:

Ими - chr_start

Адрес - 23296

Комментарий: адрес данных символа.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Эта программа полезна в играх и для серьёзных целей, например в графике.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА
            LD HL,(23296)       42     0      91
            LD E,128            30     128
N_BIT       PUSH HL             229
            LD C,0              14     0
            LD B,1              6      1
NEXT_B      LD A,E              123
            AND (HL)            166
            CP 0                254    0
            JR Z,NOT_S          40     3
            LD A,C              121
            ADD A,B             128
            LD C,A              79
NOT_S       SLA B               203    32
            INC HL              35
            JR NC,NEXT_B        48     242
            POP HL              225
            PUSH BC             197
            SRL E               203    59
            JR NC,N_BIT         48     231
            LD DE,7             17     7      0
            ADD HL,DE           25
            LD B,8              6      8
REPL        POP DE              209
            LD (HL),E           115
            DEC HL              43
            DJNZ REPL           16     251
```

```
        RET             201
```

Как она работает:

Каждый символ состоит из группы пикселей размера 8x8, каждый из которых может быть в состоянии ON (1) или OFF (0). Рассмотрим любой бит B2 байта B1 на Рис1. Данные хранятся в ячейке (B2,B1) в форме:

```
|N1   N3|
|N2   N4|
```

где: N1 = байт, в который пиксел (B2,B1) будет вставлен после вращения.
N2 = бит в N1, в который он будет вставлен.
N3 = значение, которое представляет текущее значение бита.
N4 = значение бита N2.

Каждый байт вращаемого символа будет сформирован добавлением значений всех битов N2, которые будут в новом байте.

1 128 0 1	2 64 0 1	3 32 0 1	4 16 0 1	5 8 0 1	6 4 0 1	7 2 0 1	8 1 0 1	1
1 128 1 2	2 64 1 2	3 32 1 2	4 16 1 2	5 8 1 2	6 4 1 2	7 2 1 2	8 1 1 2	2
1 128 2 4	2 64 2 4	3 32 2 4	4 16 2 4	5 8 2 4	6 4 2 4	7 2 2 4	8 1 2 4	3
1 128 3 8	2 64 3 8	3 32 3 8	4 16 3 8	5 8 3 8	6 4 3 8	7 2 3 8	8 1 3 8	4 Байт (B1)
1 128 4 16	2 64 4 16	3 32 4 16	4 16 4 16	5 8 4 16	6 4 4 16	7 2 4 16	8 1 4 16	5
1 128 5 32	2 64 5 32	3 32 5 32	4 16 5 32	5 8 5 32	6 4 5 32	7 2 5 32	8 1 5 32	6
1 128 6 64	2 64 6 64	3 32 6 64	4 16 6 64	5 8 6 64	6 4 6 64	7 2 6 64	8 1 6 64	7
1 128 7 128	2 64 7 128	3 32 7 128	4 16 7 128	5 8 7 128	6 4 7 128	7 2 7 128	8 1 7 128	8
7	6	5	4	3	2	1	0	Бит (B2)

Рис. 1 Ключ к подпрограмме вращения символа.

В HL загружается адрес первого байта символа. В регистр E загружается значение байта, который имеет 7-й бит в состоянии ON и с 0 по 6 биты - в OFF т.е. 128. HL сохраняется в стеке. В регистр C засылается 0. Далее в него будут добавляться данные, давая новое значение для формируемого байта. В регистр B загружается значение байта, нулевой бит которого включен, а биты 1-7 - выключены т.е. это единица.

В аккумулятор загружается содержимое E-регистра (3). Это значение перемножается логически (AND) с байтом, адрес которого хранится в HL. Если результат равен 0, происходит переход к NOT_S, т.к. пиксель, адресованный регистром E и регистровой парой HL, сброшен (OFF). Если же он установлен (ON), в аккумулятор загружается имеющееся значение байта (N1). Регистр B (N4) добавляется к аккумулятору, и это значение загружается в регистр C. Регистр затем устанавливается, чтобы указать на следующий байт (B1). HL увеличивается, указывая на следующий байт (B1). Если байт N1 не завершен, подпрограмма возвращается к NEXT_B.

HL восстанавливается из стека, чтобы снова указать на первый байт символа, BC

сохраняется в стеке, чтобы запомнить значение последнего байта для завершения в C регистре. E-регистр настраивается на адрес следующего бита каждого байта. Если вращение не завершено происходит переход к N_BIT.

В DE загружается значение 7, и это значение добавляется к HL. Теперь HL указывает на последний байт данных. В регистр B загружается число байтов, которые должны быть взяты из стека. Для каждого байта новое значение копируется в E, и это значение помещается в ячейку с адресом в HL. HL уменьшается, чтобы указать на следующий байт и счетчик (B-регистр) уменьшается. Если счетчик не равен 0, происходит переход к REPL.

Программа возвращается в BASIC.

6.6 Изменение атрибута.

Длина: 21

Количество переменных:2

Контрольная сумма: 1952

Назначение: Эта программа изменяет значение атрибутов всех символов экрана на задаваемое - например, могут быть изменены цвета всех символов, или весь экран может замигать и т. д.

Переменные:

Имя - data saved

Длина - 1

Адрес - 23296

Комментарий: неизменяемые биты атрибута.

Имя - new_data

Длина - 1

Адрес - 23297

Комментарий: Новые биты, вводимые в байт атрибута.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Отдельные биты атрибута каждого символа могут быть изменены с помощью инструкций AND и OR.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            LD HL,22528        33     0     88
            LD BC,768          1      0     3
            LD DE,(23296)      237    91    0      91
NEXT_B      LD A,(HL)          126
            AND E              163
            OR D               178
            LD (HL),A          119
            INC H              35
            DEC BC             11
            LD A,B             120
            OR C               177
            JK NZ,NEXT_B       32 246
            RET                201
```

Как она работает:

В пару регистров HL загружается адрес области атрибутов, а в пару регистров BC количество символов на экране, в регистр D загружается значение new_data, а в регистр E загружается значение data_saved.

В аккумулятор загружается байт с адресом, находящимся в HL, а биты устанавливаются соответственно значениям регистров D и E. Результат помещается обратно в ячейку с адресом, хранящимся в HL. HL увеличивается, указывая на следующий байт, а счетчик BC уменьшается. Если содержимое BC не равно 0, программа возвращается

к NEXT_B.

Программа возвращается в BASIC.

6.7 Смена атрибута.

Длина: 22

Количество переменных: 2

Контрольная сумма: 1825

Назначение: Эта программа ищет атрибуты с определенным значением и заменяет каждое найденное вхождение другим значением.

Переменные:

Имя - old_value

Длина - 1

Адрес - 23296

Комментарий: Значение байта, подлежащего замене.

Имя - new_value

Длина - 1

Адрес - 23297

Комментарий: Значение замещающего байта.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: Эта программа полезна для выделения областей текста и графических символов.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
           LD HL,22528        33    0     88
           LD BC,768          1     0     3
           LD DE,(23296)      237   91    0     91
NEXT_B     LD A,(HL)          126
           CP E               187
           JR NZ,NO_CH        32    1
           LD (HL),D          114
NO_CH      INC HL             35
           DEC BC             11
           LD A,B             120
           OR C               177
           JR NZ,NEXT_B       32    245
           RET                201
```

Как она работает:

В пару регистров HL загружается адрес области атрибутов, а в BC загружается число символов на экране. В E-регистр загружается old_value, а в D-регистр - new_value. В аккумулятор загружается байт, адрес которого хранится в паре HL. Если аккумулятор хранит значение, которое эквивалентно содержимому E-регистра, то в байт с адресом, имеющимся в HL, помещается содержимое D-регистра. При этом HL увеличивается, указывая на следующий байт, а счетчик BC - уменьшается. Если BC не равно 0, происходит переход к NEXT_B, иначе программа возвращается в BASIC.

6.8 Закрашивание контура.

Длина: 263

Количество переменных: 2

Контрольная сумма: 26647

Назначение: Эта программа закрашивает область экрана, ограниченную линией пикселей.

Переменные:

Имя - X_coord

Длина - 1

Адрес - 23296

Комментарий: Координата X стартовой позиции.

Имя - Y_coord

Длина - 1

Адрес - 23297

Комментарий: Координата Y стартовой позиции.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Если координата Y>175 или POINT(X,Y)=1, то программа тотчас же возвращается в BASIC.

Комментарий: Эта программа - не перемещаемая, ее стартовый адрес - 31955. Когда закрашивается большая область сложной формы, нужно большое количество свободного пространства в ОЗУ. Если это невозможно, может произойти сбой.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
           LD HL,(23296)      42    0     91
           LD A,H             124
           CP                 254   176
           RET NC             208
           CALL SUBR          205   143*125
           AND (HL)           166
           CP 0               254   0
           RET NZ             192
           LD BC,65535        1     255   255
           PUSH BC            197
RIGHT      LD HL,(23296)      42    0     91
           CALL SUBR          205   143*125
           AND (HL)           166
           CP 0               254   0
           JR NZ,LEFT         32    9
           LD HL,(23296)      42    0     91
           INC L              44
           LD (23296),HL      34    0     91
           JR NZ,RIGHT        32    236
LEFT       LD DE,0            17    0     0
           LD HL,(23296)      42    0     91
           DEC L              45
           LD (23296),HL      34    0     91
PLOT       LD HL,(23296)      42    0     91
           PUSH HL            229
           CALL SUBR          205   143*125
           OR (HL)            182
           LD (HL),A          119
           POP HL             225
           LD A,H             124
           CP 175             254   175
           JR Z,DOWN          40    44
           LD A,E             123
           CP 0               254   0
           JR NZ,RESET        32    16
           INC H              36
           CALL  SUBR         205   143*125
           AND (HL)           166
           CP 0               254   0
           JR NZ,RESET        32    7
           LD HL,(23296)      42    0     91
           INC H              36
           PUSH HL            229
```

```
              LD E,1              30     1
RESET         LD HL,(23296)       42     0      91
              LD A,E              123
              CP 1                254    1
              JR NX,DOWN          32     15
              INC H               36
              CALL SUBR           205    143*125
              AND (HL)            166
              CP 0                254    0
              JR Z,DOWN           40     6
              LD E,0              30     0
              JR DOWN             24     2
L_JUMP        JR RIGHT            24     167
DOWN          LD HL,(23296)       42     0      91
              LD A,H              124
              CP 0                254    0
              JR Z,NEXT_P         40     40
              LD A,D              122
              CP 0                254    0
              JR NZ,REST          32     16
              DEC H               37
              CALL SUBR           205    143*125
              AND (HL)            166
              CP 0                254    0
              JR NZ,REST          32     7
              LD HL,(23296)       42     0      91
              DEC H               37
              PUSH HL             229
              LD B,1              22     1
REST          LD A,D              122
              CP 1                254    1
              JR NZ,NEXT_P        32     14
              LD HL,(23296)       42     0      91
              DEC H               37
              CALL SUBR           205    143*125
              AND (HL)            166
              CP 0                254    0
              JR Z,NEXT_P         40     2
              LD D,0              22     0
NEXT_P        LD HL,(23296)       42     0      91
              LD A,L              125
              CP 0                254    0
              JR Z,RETR           40     12
              DEC L               45
              LD (23296),HL       34     0      91
              CALL SUBR           205    143*125
              AND (HL)            166
              CP 0                254    0
              JR Z,PLOT           40     129
RETR          POP HL              225
              LD (23296),HL       34     0      91
              LD A,255            62     255
              CP H                188
              JR NZ,L_JUMP        32     177
              CP 1                189
              JR NZ,L_JUMP        32     174
              RET                 201
SUBR          PUSH BC             197
              PUSH DE             213
              LD A,175            62     175
              SUB H               148
              LD H,A              103
              PUSH HL             229
              AND 7               230    7
              ADD A,64            198    64
```

```
        LD C,A          79
        LD A,H          124
        RRA             203   31
        RRA             203   31
        RRA             203   31
        AND 31          230   31
        LD B,A          71
        AND 24          230   24
        LD D,A          87
        LD A,H          124
        AND 192         230   192
        LD E,A          95
        LD H,C          97
        LD A,L          125
        RRA             203   31
        RRA             203   31
        RRA             203   31
        AND 31          230   31
        LD L,A          111
        LD A,E          123
        ADD A,B         128
        SUB D           146
        LD E,A          95
        LD D,0          22    0
        PUSH HL         229
        PUSH DE         213
        POP HL          225
        ADD HL,HL       41
        ADD HL,HL       41
        ADD HL,HL       41
        ADD HL,HL       41
        ADD HL,HL       41
        POP DE          209
        ADD HL,DE       25
        POP DE          209
        LD A,E          123
        AND 7           230   7
        LD B,A          71
        LD A,8          62    8
        SUB B           144
        LD B,A          71
        LD A,1          62
ROTATE  ADD A,A         135
        DJNZ ROTATE     16    253
        RRA             230   31
        POP DE          209
        POP BC          193
        RET             201
```

Как она работает:

Эта программа вычерчивает горизонтальные линии из смежных пикселей. Назовем их строчками. Предел заполнения области строчками ограничен включенными (ON) пикселями. Каждая строчка запоминается с помощью занесения в стек координат крайнего правого пикселя этой строчки.

Запускаясь с определенных координат, программа делает заполнение в каждой строчке, отмечая позиции каждой из невыполненных строчек выше или ниже. По завершении одной строчки последнее значение отмеченных координат восстанавливается и для соответствующей строчки происходит заполнение. Этот процесс повторяется до тех пор, пока не останется незаполненных строчек.

Рис. 2 иллюстрирует технику процесса. Квадраты представляют пиксели, X обозначает стартовую позицию в пределах области штриховки, а * обозначает крайние правые пиксели строчек.

////	////	////	////	////	////	////	////
////	o	////		*	////	////	////
////	o	////		X			////
////	////	////				*	////
////			*	////		*	////
////		*	////	////	////	*	////
////		*	////	////	////	*	////
////	////	////	////	////	////	////	////

Рис. 2 Иллюстрация техники заполнения области. X - это стартовая позиция, * - начало строчек, o - оставшаяся незаштрихованная область.

Программа штрихует горизонтальную линию, содержащую стартовую позицию и сохраняет в стеке позицию начала строчки на линиях непосредственно выше и ниже. Далее она штрихует линию выше, а затем ниже, отмечая в последнем случае, что ещё 2 строчки запускаются на следующей нижней строке и т.д. Любая позиция в пределах области для штриховки может быть выбрана как стартовая позиция. Но заметим, что два пиксела, промаркированные нулями, нетронуты, т.к. они отделены от заштриховываемой области.

В регистр H загружается Y-координата, а в L-регистр - X-координата. Если значение Y больше, чем 175, программа возвращается в BASIC. Процедура SUBR вызывается, возвращая адрес бита (X,Y) в память. Если этот бит в состоянии 'ON' (включен), подпрограмма возвращается в BASIC.

Число 65535 помещается в стек, чтобы отметить первое сохраненное значение. Позднее, когда число восстанавливается из стека, оно интерпретируется, как пара координат. Однако, если число равно 65535, происходит возврат в BASIC, т.к. программа закончена.

В регистр H загружается Y-координата, а в регистр L - X координата. Процедура SUBR вызывается, возвращая в HL адрес бита (X,Y). Если этот бит установлен (ON), происходит переход к LEFT. Иначе X-координата увеличивается, и делается переход к RIGHT, если X не равен 256.

В процедуре LEFT, DE устанавливается в 0. Регистры D и E используются, как флаги: D - вниз (DOWN), E - вверх (UP). X-координата уменьшается. SUBR вызывается, и вычерчивается точка (X,Y). Если Y-координата равна 175, подпрограмма переходит к DOWN. Если флаг "ВВЕРХ" установлен, происходит переход к RESET. Если бит (X,Y+1) сброшен, значение X и Y+1 сохраняются в стеке и флаг "ВВЕРХ" включается.

В процедуре RESET, если флаг "ВВЕРХ" включен, происходит переход к DOWN. Если бит (X,Y+1) включен (ON). Флаг "ВВЕРХ" выключается. В процедуре DOWN, если Y-координата равна 0, происходит переход к NEXT_PIXEL. Если флаг "ВНИЗ" включен, происходит переход к REST. Если бит (X,Y-1) сброшен (OFF), то значения X и Y-1 сохраняются на стеке и флаг "ВНИЗ" включается.

В процедуре REST, если флаг "ВНИЗ" выключен, происходит переход к NEXT_P. Если бит (X,Y-1) установлен (ON), то флаг "ВНИЗ" выключается. В процедуре NEXT_P, если X-координата равна 0, подпрограмма переходит к RETR. X-координата уменьшается, и, если новый бит (X,Y) сброшен (OFF), происходит переход к PLOT. В процедуре RETR X и Y-координаты извлекаются из стека. Если X и Y равны 255, то - возврат в BASIC, т.к.

заполнение области завершено. Иначе подпрограмма возвращается к RIGHT.

Процедура SUBR должна подсчитать адрес бита (X,Y) в памяти. В BASIC этот адрес будет:

16384 + INT(Z/8) + 256 * (Z - 8 * INT(Z/8)) + 32 * (64 * INT(Z/64) + INT(Z/8) - 8 * INT(Z/64)), где Z=175-Y

Пары регистров BC и DE сохраняются на стеке. В аккумулятор засылается число 175 и из этого значения вычитается Y-координата. Результат копируется в H-регистр. Затем HL сохраняется на стеке. Пять левых битов аккумулятора устанавливаются в 0, а затем к ним прибавляется 64. Результат копируется в C-регистр. При умножении на 256 получаем:

16384 + 256 * (Z-8 * INT(Z/8)).

В аккумулятор загружается Z, это значение делится на 8, результат копируется в регистр B. Этот результат - INT(Z/8). Установка трех крайних правых битов в 0 при ротации дает значения 8 * INT(Z/64), которое загружается в D-регистр. В аккумулятор загружается Z и 6 крайних правых битов выключаются, что дает 64 * INT(Z/64). Это значение загружается в E-регистр. Значение из C-регистра копируется в H. В аккумулятор загружается X-координата, это значение делится на 8, а результат копируется в L.

В аккумулятор затем загружается значение E-регистра и к нему прибавляется содержимое B. Значение D-регистра вычитается и результат загружается в DE. Это значение умножается на 32. DE восстанавливается из стека и прибавляется к HL. Т.о. HL теперь хранит адрес бита (X,Y).

В аккумулятор загружается первоначальное значение X. Установка пяти левых битов в ноль дает значение X - 8 * INT(X/8). В B регистр затем загружается 8 минус значение аккумулятора, чтобы использовать его в качестве счетчика.

Аккумулятор устанавливается в 1, и это умножается на 2 (B-1) раз.

В этот момент в аккумуляторе необходимо установить бит, который соответствует биту (X,Y) с адресом в HL. DE и BC затем восстанавливаются из стека, и SUBR выполняет возврат в основную программу.

6.9 Построение шаблонов.

Длина: 196

Количество переменных:2

Контрольная сумма: 20278

Назначение: Эта программа чертит шаблон любого размера на экране. Под шаблоном понимается любая, ранее определенная фигура.

Переменные:

Имя - X_start

Длина -1

Адрес - 23296

Комментарий: X-координата первого пиксела.

Имя Y_start

Длина -1

Адрес - 23297

Комментарий: Y-координата первого пиксела.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Если строковая переменная, хранящая информацию по шаблону A$, не существует, имеет нулевую длину или не содержит никакой информации, программа возвращается непосредственно в BASIC. Это происходит также в случае, если Y_start больве, чем 175.

Комментарий: Это полезная программа для хранения фигур в памяти и быстрого вычерчивания их на экране.

Использование этой программы:

(I) LET A$="информация по шаблону"

(II) POKE 23296, X-координата первого пиксела

(III)POKE 23297, Y-координата первого пиксела

(IV) RANDOMIZE USR адрес

Информация шаблона - это символов, который имеет следующий формат:
" 0 " поместить точку
" 5 " уменьшить X-координату
" 6 " уменьшить Y- координату
" 7 " увеличить X-координату
" 8 " увеличить Y координату
Любые другие символы игнорируются

Программа включает в себя возможность 'wrap-round'. Т.е. если X-координата выходит за левую часть экрана, шаблон появляется справа и т.п. Чтобы изменить программу для использования иной строковой переменной вместо A$, нужно изменить 65 * (код буквы A) на код иного символа.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР         ЧИСЛА ДЛЯ ВВОДА
            LD HL,(23627)     42    75    92
NEXT_V      LD A,(HL)         126
            CP 128            254   128
            RET Z             200
            BIT 7,A           203   127
            JR NZ,FORNXT      32    23
            CP 96             254   96
            JR NC,NUMBER      48    11
            CP 65             254   65*
            JR Z,FOUND        40    35
STRING      INC HL            35
            LD E,(HL)         94
            INC HL            35
            LD D,(HL)         86
ADD         ADD HL,DE         25
            JR INCR           24    5
NUMBER      INC HL            35
            INC HL            35
            INC HL            35
            INC HL            35
            INC HL            35
INCR        INC HL            35
            JR NEXT_V         24    225
FORNXT      CP 224            254   224
            JR C,N_BIT        56    5
            LD DE,18          17    18    0
            JR ADD            24    236
N_BIT       BIT   5,A         203   111
            JR Z,STRING       40    228
NEXT_B      INC HL            35
            BIT 7,(HL)        203   126
            JR Z,NEXT_B       40    251
            JR NUMBER         24    228
FOUND       INC HL            35
            LD C,(HL)         78
            INC HL            35
            LD B,(HL)         70
            INC HL            35
            EX DE,HL          235
            LD A,(23297)      58    1     91
            CP 176            254   176
            RET NC            208
AGAIN       LD HL,(23296)     42    0     91
            LD A,B            120
            OR C              177
```

```
         RET Z            200
         DEC BC           11
         LD A,(DE)        26
         INC DE           19
         CP 48            254   48
         JR NZ,NOT_PL     32    78
         PUSH BC          197
         PUSH DE          213
         LD A,175         62    175
         SUB H            148
         LD H,A           103
         PUSH   HL        229
         AND 7            230   7
         ADD A,64         198   64
         LD C,A           79
         LD A,H           124
         RRA              203   31
         RRA              203   31
         RRA              203   31
         AND 31           230   31
         LD B,A           71
         AND 24           230   24
         LD D,A           87
         LD A,H           124
         AND 192          230   192
         LD E,A           95
         LD H,C           97
         LD A,L           125
         RRA              203   31
         RRA              203   31
         RRA              203   31
         AND 31           230
         LD L,A           111
         LD A,E           123
         ADD A,B          128
         SUB D            146
         LD E,A           95
         LD D,0           22    0
         PUSH HL          229
         PUSH DE          213
         POP HL           225
         ADD HL,HL        41
         ADD HL,HL        41
         ADD HL,HL        41
         ADD HL,HL        41
         ADD HL,HL        41
         POP DE           209
         ADD HL,DE        25
         POP DE           209
         LD A,E           123
         AND 7            230   7
         LD B,A           71
         LD A,8           62    8
         SUB B            144
         LD B,A           71
         LD A,L           62    1
ROTATE   ADD A,A          135
         DJNZ ROTATE      16    253
         RRA              203   31
         POP DE           209
         POP BC           193
         OR (HL)          182
         LD (HL),A        119
HERE     JR AGAIN         24    165
NOT_PL   CP 53            254   53
```

```
            JR NZ,DOWN         32     1
            DEC 1              45
DOWN        CP 54              254    54
            JR NZ,UP           32     8
            DEC H              37
            LD H,A             124
            CP 255             254    255
            JR NZ,SAVE         32     19
            LD H,175           38     175
UP          CP 55              254    55
            JR NZ,RIGHT        32     8
            INC H              36
            LD A,H             124
            CP 176             254    176
            JR NZ,SAVE         32     7
            LD H,0             38     0
RIGHT       CP 56              254    56
            JR NZ,SAVE         32     1
            INC L              44
SAVE        LD (23296),HL      34     0
            JR HERE            24     215
```

Как она работает:

Для нахождения адреса строковой переменной используется немного измененная первая часть программы 'Поиск подстроки'.

Длина строковой переменной загружается в BC, а адрес первого символа A$ загружается в DE. В аккумуляторе устанавливается начальное значение Y, и, если оно больше 175, подпрограмма возвращается в BASIC. В H регистр загружается Y-координата, а в L X-координата. Если значение пары регистров BC равно 0, подпрограмма возвращается в BASIC, т.к. достигнут конец строковой переменной. BC уменьшается, чтобы показать, что обрабатывается следующий символ. Следующий символ загружается в аккумулятор и DE увеличивается, указывая на следующий байт. Если аккумулятор не содержит код 48, происходит переход к NOT_PL. Точка (X,Y) вычерчивается используя процедуру SUBR из программы "Закрашивание контура". Затем программа возвращается назад к 'AGAIN'. В процедуре NOT_PL, если аккумулятор содержит число 53, X-координата уменьшается. В процедуре DOWN, если аккумулятор не содержит число 54, делается переход к UP. Y-координата уменьшается, и, если ее значение становится равным -1, Y-координата устанавливается на значение 175.

В процедуре UP, если аккумулятор не содержит 55, происходит переход к RIGHT. Y-координата увеличивается, и, если она равна 176, то Y-координата устанавливается в 0. В процедуре RIGHT, если аккумулятор содержит значение 56, X-координата увеличивается. В процедуре SAVE координаты X и Y помещаются в память, а программа делает переход к HERE.

6.10 Увеличение экрана и копирование.

Длина:335

Количество переменных:8

Контрольная сумма: 33663

Назначение: Эта программа копирует часть дисплея в другую область экрана, увеличивая копию по X или по Y.

Переменные: см. рис.3

Имя	Длина	Адрес	Комментарий
upper_Y_co-ord	1	23296	Y-координата верхнего ряда
lower_Y_co-ord	1	23297	Y-координата нижнего ряда
right_X_co-ord	1	23298	X-координата крайней правой колонки
left_X_co-ord	1	23299	X-координата крайней левой колонки
horizontal_scale	1	23300	Увеличение по X

vertical_scale	1	23301	Увеличение по Y
new_left_co-ord	1	23302	X-координата крайней левой колонки области, в которую делается копирование
nev_lower_co-ord	1	23303	Y-координата нижнего ряда области, в которую делается копирование

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Программа возвращается в BASIC, если одно из условий верно.

(I) horizontal_scale=0

(II) vertical_scale=0

(III) upper_Y_co-ord больше, чем 175

(IV) new_lower_co-ord больше, чем 175

(V) lower_Y_co-ord больше, чем upper_co-ord

(VI) new_lower_co-ord больше, чем right_X_co-ord

Однако, для краткости программы нет контроля, который проверял бы возможность размещения новой картинки на экране. Если этого не получается, может произойти сбой. Программа также требует большого объема свободной области ОЗУ, и, если это не доступно, может произойти сбой.

Комментарий: Эта программа - не перемещаемая из-за процедуры PLOT. Она размещается по адресу 65033 и, если скопированная область экрана имеет тот же самый размер, что и оригинал, масштаб должен быть установлен в 1, для двойного размера загружается масштаб 2:1, для тройного размера загружается масштаб 3:1 и т.д.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
           LD IX,23296        221   33    0     91
           LD A,175           62    175
           CP (IX+0)          221   190   0
           RET C              216
           CP (IX+7)          221   190   7
           RET C              216
           SUB A              151
           CP (IX+4)          221   190   4
           RET Z              200
           CP (IX+5)          221   190   5
           RET Z              200
           LD HL,(23296)      42    0     91
           LD B,L             69
           LD A,L             125
           SUB H              148
           RET C              216
           LD (23298),A       50    0     91
           LD E,A             95
           LD HL,(23298)      42    2     91
           LD C,L             77
           LD A,L             125
           SUB H              148
           RET C              216
           LD (23298),A       50    2     91
           PUSH BC            197
           LD L,A             111
           LD H,0             38    0
           INC HL             35
           PUSH HL            229
           POP BC             193
           INC E              28
ADD        DEC E              29
           JR Z,REMAIN        40    3
```

```
            ADD HL,BC          9
            JR ADD             24    250
REMAIN      LD A,L             125
            AND 15             230   15
            LD B,A             71
            POP HL             225
            LD C,L             77
            JR NZ,SAVE         32    2
FULL        LD B,16            6           16
SAVE        PUSH HL            229
            CALL SUBR          205   13    255
            AND (HL)           166
            JR Z,OFF           40    2
            LD A,1             62    1
OFF         POP HL             225
            RRA                203   31
            RL E               203   19
            RL D               203   18
            LD A,L             125
            CP (IX+3)          221   190   3
            JR Z,NEXT_R        40    6
            DEC L              45
N_BIT       DJNZ SAVE          16    231
            PUSH DE            213
            JR FULL            24    226
NEXT_R      LD L,C             105
            LD A,H             124
            CP (IX+1)          221   190   1
            JR Z,COPY          40    3
            DEC H              37
            JR N_BIT           24    241
COPY        PUSH DE            213
            LD B,0             60
            LD H,B             96
            LD L,B             104
RESET       LD (23306),HL      34    10    91
            LD A,B             120
            OR A               183
            JR NZ,RETR         32    3
            POP DE             209
            LD B,16            6     16
RETR        SUB A              151
            DEC B              5
            RR D               203   26
            RR E               203   27
            RL A               203   23
            PUSH DE            213
            PUSH BC            197
            PUSH AF            245
            LD H,1             38    1
LOOP        LD L,1             46    1
PRESET      LD (23304),HL      34    8     91
            LD A,(23307)       58    11    91
            LD HL,0            33    0     0
            LD DE,(23301)      237   91    5     91
            LD D,L             85
MULTIP      OR A               183
            JR Z,CALC          40    6
            ADD HL,DE          25
            DEC A              61
            JR MULTIP          24    249
L_JUMP      JR RESET           24    208
CALC        LD A,(23303)       58    7     91
            ADD A,L            133
            LD HL,(23304)      42    8     91
```

```
            ADD A,L             133
            DEC A               61
            PUSH AF             245
            LD A,(23306)        58     10     91
            LD HL,0             33     0      0
            LD DE,(23300)       237    91     4      91
            LD D,L              85
REPEAT      OR A                183
            JR Z,CONTIN         40     4
            ADD HL,DE           25
            DEC A               61
            JR REPEAT           24     249
CONTIN      LD A,(23302)        58     6      91
            ADD A,L             133
            LD HL,(23305)       42     9      91
            ADD H,L             133
            DEC A               61
            LD L,A              111
            POP AF              241
            LD H,A              103
            POP AF              241
            PUSH AF             245
            OR A                183
            JR NZ,PLOT          32     7
            CALL SUBR           205    13     255
            CPL                 47
            AND (HL)            166
            JR POKE             24     4
PLOT        CALL SUBR           205    13     255
            OR (HL)             182
POKE        LD (HL),A           119
            LD HL,(23304)       42     8      91
            INC L               44
            LD A,(23301)        58     5      91
            INC A               60
            CP L                189
            JR NZ,PRESER        32     165
            INC H               36
            LD A,(23300)        58     4      91
            INC A               60
            CP H                188
            JR NZ,LOOP          32     155
            POP AF              241
            POP BC              193
            POP DE              209
            LD HL,(23306)       42     10     91
            INC L               44
            LD A,(23298) 58     2      91
            INC A               60
            CP L                189
            JR NZ,L_JUMP        32     164
            LD L,0              46     0
            INC H               36
            LD A,(23296)        58     0      91
            INC A               60
            CP H                188
            JR NZ,L_JUMP        32     154
            RET                 201
SUBR        PUSH BC             197
            PUSH DE             213
            LD A,175            62     175
            SUB H               148
            LD H,A              103
            PUSH HL             229
            AND 7               230    7
```

```
         ADD A,64           198  64
         LD C,A             79
         LD A,H             124
         RRA                203  31
         RRA                203  31
         RRA                203  31
         AND 31             230  31
         LD B,A             71
         AND 24             230  24
         LD D,A             87
         LD A,H             124
         AND 192            230  192
         LD E,A             95
         LD H,C             97
         LD A,L             125
         RRA                203  31
         RRA                203  31
         RRA                203  31
         AND 31             230  31
         LD L,A             111
         LD A,E             123
         ADD A,B            128
         SUB D              146
         LD E,A             95
         LD D,0             22   0
         PUSH HL            229
         PUSH DE            213
         POP HL             225
         ADD HL,HL          41
         ADD HL,HL          41
         ADD HL,HL          41
         ADD HL,HL          41
         ADD HL,HL          41
         POP DE             209
         ADD HL,DE          25
         POP DE             209
         LD A,E             123
         AND 7              230  7
         LD B,A             71
         LD A,8             62   8
         SUB B              144
         LD B,A             71
         LD A,1             62   1
ROTATE   ADD A,A            135
         DJNZ ROTATE  16    253
         RRA                203  31
         POP DE             209
         POP BC             193
         RET                201
```

Как она работает:

В IX загружается адрес буфера принтера для использования его в качестве указателя переменных. Если верхняя Y-координата или новая нижняя координата больше 175, программа возвращается в BASIC. Если значение увеличения по горизонтали или вертикали равно 0, происходит возврат в BASIC.

В H-регистр загружается нижняя Y-координата, а в L-регистр - верхняя Y- координата. L-регистр копируется в B-регистр и в аккумулятор. H-регистр вычитается из аккумулятора и подпрограмма возвращается в BASIC, если результат отрицательный.

Значение аккумулятора затем помещается в ячейку 23298 для использования в качестве счетчика. Пара регистров BC затем сохраняется на стеке.

Регистр HL загружается значением аккумулятора, увеличивается и копируется в регистр BC. BC прибавляется к HL E раз, результирующее значение в HL является числом пикселей для копирования. В аккумулятор загружается значение L-регистра и 4 крайних

левых бита устанавливаются в 0. Результат копируется в В-регистр для использования его в качестве счетчика.

Пара регистров HL восстанавливается из стека и регистр L копируется в С-регистр. Если В-регистр содержит 0, в регистр загружается число 16 - это количество битов в регистровой паре. Затем вызывается процедура SUBR и в аккумулятор загружается значение POINT (L,H). Пара регистров DE сдвигается влево, а значение бита из аккумулятора загружается в крайний правый бит Е- регистра.

Если L-регистр равен левой X-координате, подпрограмма переходит к NEXT_R (следующий ряд). Иначе уменьшается L-регистр, а затем - В-регистр. Если В-регистр не содержит 0, подпрограмма возвращается к SAVE для подачи следующего бита в пару регистров DE. Если В-регистр содержит 0, пара регистров DE помещается в стек и происходит переход к FULL.

В процедуре NEXT_R в L-регистр загружается правая X-координата, а в аккумулятор загружается значение Н-регистра. Если значение аккумулятора равно нижней Y-координате, происходит переход к COPY, т.к. последний пиксель для копирования подан в DE. Иначе, Н-регистр уменьшается, указывая на следующий ряд, и подпрограмма возвращается к N_BIT.

В процедуре COPY содержимое пары ВЕ помещается в стек, а В, Н и L-регистры устанавливаются в 0 для использования их в качестве счетчиков. Содержимое пары регистров HL помещается в ячейку с адресами 23306/7 - HL теперь может использоваться как счетчик последующих циклов без использования стека. Если В-регистр содержит 0, DE восстанавливается из стека, а регистр В повторно устанавливается на значение 16, определяя количество пикселей, хранимых в DE. В-регистр уменьшается показывая, что бит информации удален из DE. Крайний правый бит Е-регистра загружается в аккумулятор, а пара регистров DE сдвигается вправо. DE,BC и AF помещаются в стек до тех пор, пока выполнятся определенные расчеты.

В регистры Н и L загружается 1 для использования их в качестве счетчика, а HL помещается в ячейки с адресами 23304/5. В аккумулятор загружается значение байта по адресу 23307 - это один из счетчиков, сохраненных ранее. В пару регистров DE загружается масштаб по вертикали. Это значение затем умножается на значение аккумулятора, а результат подаётся в HL. Это значение прибавляется к новой нижней Y-координате в аккумуляторе. Байт по адресу 23304 затем добавляется к аккумулятору, а результат уменьшается.

Аккумулятор теперь хранит Y-координату для построения следующего пиксела. Это значение хранится в стеке до тех пор, пока подсчитывается X-координата похожим способом. После расчета X-координата загружается в L-регистр. Y-координата восстанавливается из стека и загружается в Н-регистр. В аккумуляторе устанавливается последнее значение, хранящееся в стеке. Если оно равно 1, то должна быть построена точка (X,Y), иначе должна быть выполнена процедура UNPLOTTED. Вызывается SUBR и выполняются соответствующие действия.

Пара регистров HL загружается счетчиками цикла, хранящимися адресам 23304/5. Регистр L увеличивается, и, если он не содержит значение (1+vertical_scale), программа возвращается к PRESER. Регистр Н увеличивается, и, ее он не содержит значение (1+horizontal_scale), происходит переход к LOOP.

Пары регистров AF,BC и DE восстанавливаются из стека, а в пару регистров HL загружается второе значение набора счетчиков цикла, которое хранится по адресам 23306/7. Регистр L увеличивается и происходит переход к RESET, если результат не равен (right_X_co-ord-left_X_co-ord + 1)

Регистр L устанавливается в 0 - это первоначальное значение счетчика цикла. Н-регистр затем увеличивается и подпрограмма переходит к RESET, если результат не равен (upper_Y_co-ord -lower_Y_co-ord + 1). Программа возвращается в BASIC,

Процедура SUBR идентична той, что используется в программе "Закрашивание контура".

Продолжение следует

Алексеев А. Г.

МАСТЕРФАЙЛ-09 Полная русификация

Окончание.
Начало ZX-РЕВЮ-92',стр-29-32.

Текстовые сообщения, выводимые на экран в любой программе кроме печатаемых символов содержат также управляющие коды AT, TAB, INK, PAPER и т.д. Их без крайней необходимости лучше не изменять, так как это может нарушить работу программы, но в некоторых случаях придётся изменять и их. Для этого в программе мониторе предусмотрена строка 231. Если Вы уберёте REM из строки 231 и подставите REM в начало строки 230, то работа программы несколько изменится. Будет такой же вывод на экран, так же запрашивается информация, но теперь без кавычек. Это означает что ожидается ввод кода - числа от 0 до 255, которое после нажатия "ENTER" непосредственно будет записано в память. Режим ввода кодов будет нужен редко, но все-таки иногда пригодится. Для того, чтобы остановить программу в этом режиме, надо просто нажать "КУРСОР ВНИЗ" (CAPS SHIFT+6) или нажать "STOP" (SYMBOL SHIFT + A) и "ENTER".

Теперь подставьте REM в обе строки 230 и 231 и сделайте RUN. После ввода адреса Вам будет выдан дамп памяти. Этот режим Вы будете использовать для поиска текстовых сообщений в программе. С этого момента можно начинать работу, но прежде - несколько примеров.

Введите адрес 60350. Вы увидите на экране:

```
60350        237     GO SUB
60351        225     LLIST
60352        201     <>
60353        22      ?
60354        2       ?
60355        0       ?
60356        17      ?
60357        6       ?
60358        86      V
60359        69      E
60360        82      R
60361        84      T
60362        73      I
60363        67      C
60364        65      A
60365        76      L
60366        32
60367        76      L
60368        73      I
60369        78      N
60370        69      E
60371        255     COPY
```

В ячейках 60350 находится какая - то программа в машинных кодах, а вот с ячейки 60353 начинается строка символов. При этом в ячейках 60353…60357 находятся управляющие символы, а сам текст расположен в ячейках 60358…60370. Потом идет символ с кодом 255. который завершает строку текста. Далее опять начинается программа в машинных кодах.

Управляющие символы имеют следующее значение:

Код: 16 - упр. INK
Код: 17 - упр. PAPER
Код: 18 - упр. FLASH

Код: 19 - упр. BRIGHT
Код: 20 - упр. INVERSE
Код: 21 - упр. OVER
Код: 22 - упр. AT
Код: 23 - упр. TAB

Так что комбинация кодов в ячейках 60353...60358 эквивалентна фрагменту Бейсик-строки:

```
... AT 2,0; PAPER  6; ...
```

В данном случае нас интересует только текст, который должен быть заменен:

```
VERTICAL  LINE
ВЕРТИК.  ЛИНИЯ
```

(Можете, убрав REM из строки 230, запустить программу, задать адрес 60358 и попробовать заменить текст на русский.)

Другой пример. После старта программы-монитора задайте адрес 58425. В дампе памяти Вы узнаете текст основного меню MF 09:

```
AADD A RECORD ...
CCHOOSE A REPORT
DDISPLAY/PRINT ..
     . . .
```

Правда он при работе программы выводится несколько иначе:

```
ADD A  RECORD .......A
CHOOSE A REPORT .....C
DISPLAY/PRINT .......D
     . . .
```

В памяти сначала идет символ той клавиши, которая должна быть нажата, затем текст сообщения. Вдаваться в детали работы программы нет необходимости. У нас другая задача. Оставив без изменения символы тех клавишей, которые должны нажиматься (первые буквы, они набраны в режиме курсора [C], надо заменить текст на русский:

```
АНОВАЯ ЗАПИСЬ...
СОБЗОР ФОРМАТОВ.
ОДИСПЛЕЙ/ПЕЧАТЬ.
     . . .
```

Еще пример. После старта монитора задайте адрес 59759. На экране увидите:

```
59759   78   N
59760   79   O
59761   32
59762   84   T
59763   73   I
59764   84   T
59765   76   L
59766   197  OR
```

При работе программы эта надпись выводится так:

```
NO TITLE
```

Откуда же берется последняя буква <E> ?

Вообще надо сказать, что перед тем, как в машинных кодах подана команда вывода строки символов на экран, предварительно задан адрес начальной ячейки, где расположена строка символов и длина этой строки. Анализ программы MF 09 показывает, что здесь применяется еще и другой способ вывода. Длина выводимой строки не указывается, однако во время вывода анализируется код выводимого символа. Если это код с 0 по 127, то продолжается вывод на экран, а если код символа больше или равен 128, то это значит, что процедура вывода этим символом заканчивается, при этом на экран выводится символ, код которого на 128 меньше, чем содержащийся в памяти. В ячейке 59766 стоит OR. Его код (читаем на экране) равен 197. 197-128=69. По таблице кодов "Спектрума", а если ее нет под

рукой, то сделав PRINT CHR$ 69, выясняем, что этому коду соответствует буква "E". Вот откуда взялась последняя буква.

Этот текст можно заменить следующим образом:

```
NO TITLE
НЕ НАЗВ.
```

При этом буквы "НЕ НАЗВ" заменяем непосредственно русскими буквами, а вместо точки, код которой (выясняем: PRINT CODE ".") равен 46, надо ввести символ с кодом 46+128=174. По таблице кодов "Спектрума" или сделав PRINT CHR$ 174, определяем, что это VAL$. Можно ввести его в том же режиме, нажав "EXT. MODE", затем "SYMBOL SHIFT"+"j". Часто придется вводить в качестве последнего символа - "пробел" (его код 32). Это будет символ, код которого равен 32+128 =160. По таблице кодов находим, что это символ "Q" UDG-графики. Вводя его, нажмите "GRAPH" (CAPS SHIFT+9), затем "Q", затем еще раз "GRAPH" (пусть Вас не смущает, что напечатается что-то непонятное, так как на месте символов UDG-графики находятся коды программы MF 09). Однако, могут попадаться такие символы, которые с клавиатуры вводятся в режиме курсора [K]. Например, начиная с адреса 64522 находится текст:

```
64522   65     A
64523   82     R
64524   71     G
64525   32
64526   78     N
64527   79     O
64528   84     T
64529   32
64530   78     N
64531   85     U
64532   77     M
64533   69     E
64534   82     R
64535   73     I
64536   195    NOT
```

В ячейке 64536 "спрятана" буква, код которой 195-128=67. Это латинская буква "C". Заменяем эту строку на русский текст:

```
ARC NOT NUMERIC
АРГ.НЕ ЧИСЛОВОЙ
```

При этом вместо последней русской буквы "И", (ее код выясняем, сделав PRINT CODE "И" - при этом "И" набрана в режиме курсора [L], так как включен русско-латинский символьный набор; он равен 106) надо ввести символ с кодом 106+128=234. Это REM. Ввести этот символ, находясь в режиме курсора [L] или [C] никак не удастся. Для этого надо переключить курсор на [K] (командный режим). Как это сделать? Введите ключевое слово "THEN" (SYMBOL SHIFT+G). Теперь курсор стал [K] и Вы можете ввести "REM", нажав "E". Теперь нажмите "КУРСОР ВЛЕВО" и удалите "THEN" при помощи "DELETE" (CAPS SHIFT+0). Далее - "ENTER" - ввод кода в память.

Можно это сделать и иначе - просто остановить программу и "вручную" сделать POKE 64536,234.

Теперь Вы знаете, как располагаются текстовые сообщения в программе MF 09 и умеете работать с программой-монитором. Подставьте REM в обе строки 230 и 231 и начинайте работу. Запустив программу-монитор, задайте адрес 57328. (С адреса 56560 по 57327 расположен символьный набор, там делать нечего.) Сейчас задача - просмотреть все коды программы от начала до конца, найти все текстовые сообщения в программе и

записать их адреса и сами сообщения на бумагу, оставляя место для последующего перевода.

Далее, вооружившись англо-русским словарем, начинаем перевод. Не надо стремиться переводить текст дословно, так как мы слишком сильно сжаты рамками того места, которое отведено в программе под ту или иную фразу. Более подойдет литературный перевод, при этом, конечно, лучше, если Вы уже достаточно поработали с программой MF 09 и Вам понятен смысл переводимых сообщений. Это позволит Вам найти подходящую по смыслу замену, даже если она не является переводом английской фразы. Что касается программы MF 09, то сообщения, которые Вы там встретите и их перевод, реализованный в моем варианте "MF09 RUS", приведен ниже. Когда же Вы возьметесь за какую-нибудь другую программу, то Вам придется проделать самим всю эту работу. Это, пожалуй, самая большая по затратам времени, ответственная и творческая часть работы. Так что запаситесь терпением, может быть не на один день. Остальное - дело чисто механическое.

Текстовые сообщения программы MF 09 и их перевод

Вначале каждого сообщения указан адрес, с которого необходимо произвести замену текста. Далее - английский текст и под ним русский вариант перевода.

Подчеркивающая черта обозначает "пробел".

Символы, коды которых должны быть на 128 больше, отмечены "*" в конце строки, далее идет код символа и сам символ или ключевое слово.

```
57764:
ITEM_ALREADY_IN_RECORD  * 196 BIN
ЭТО_ПОЛЕ_УЖЕ_ВВЕДЕНО__  * 160
                        Q-GRAPH
57787:
AADD_ITEM........ RREPLACE_ITEM...
AНОВОЕ_ПОЛЕ...... RИЗМЕНЕНИЕ ПОЛЯ.

EERASE_ITEM...... HNEXT_ITEM......
EУДАЛЕНИЕ_ПОЛЯ... HВЫБОР_ПОЛЯ.....

DDISPLAY/PRINT... GGET_ITEM.......
DДИСПЛЕИ/ПЕЧАТЬ.. GВЫЗОВ_ПОЛЯ.....

PPROMPT_ITEMS... =ANOTHER,RECORD.
PАВТОЗАПРОС..... =СЛЕДУЮЩАЯ ЗАП..

MMAIN_MENU......
MГЛАВНОЕ_МЕНЮ...

57940:
ENTER_TEXT_1-128_CHARS.
ВВОД_ТЕКСТА_1-128_СИМВ.

57988:
GIVE_DATA_REF * 198 AND
МЕТКА_ДАННЫХ_ * 160 Q-GRAPH

58089:
NOT_NAMED
НЕТ_ИМЕНИ

58130:
SAVE_PROG/FILE.
ЗАПИСЬ.ПР./ФАЙЛ

58243:
Y-TO_CONFIRM * 205 STEP
```

```
Y-ЕСЛИ_ДА___ * 160 Q-GRAPH

58425:
AADD_A_RECORD... CCHOOSE_A_REPORT
AНОВАЯ_ЗАПИСЬ... CОБЗОР_ФОРМАТОВ.

DDISPLAY/PRINT.. EEDIT_FORMAT_DEF
DДИСПЛЕИ/ПЕЧАТЬ. EРЕДАКТ._ФОРМАТА

LLOAD_A_FILE.... NNAME_DATA_REF..
LЗАГРУЗКА_ФАЙЛА. NИМЕНА_ПОЛЕЙ_ДАН

SSEARCH_THE_FILEIIHVERT_SELECTN.
SПОИСК.......... ИНВЕРСИЯ_ВЫБОРА

RRESET_SELECTIONPPURGE_SEL_RECDS
RСБРОС_ВЫБОРА...PУДАЛЕН.ВЫБР.ЗАП

TTOTAL/AVERAGE.. VSAVE_PROG/FILE
TИТОГ_ОБЩ./СРЕДНVЗАПИСЬ_ПР./ФАЙЛ

UEXEC_USER_BASIC
UВЫПОЛНИТЬ_BASIC

58903:
FILE_ONLY.......... F
ТОЛЬКО_ДАННЫЕ...... F

58998:
_RECS=00000_SEL=00000_SPA=00000_
__ЗАП=00000_ВЫБ=00000_СВБ=00000_

59326:         59759:
FILE_FULL      NO_TITLE  * 197 OR
НЕТ_МЕСТА      НЕ_НАЗВ.  * 174 VAL$

60065:         60077:
DELETE         COPY
УДАЛ._         КОП.

60082:
ALREADY_DEFINED  * 196 BIN
МЕТКА_ЗАНЯТА__   * 160 Q-GRAPH

60097:
NO_SUCH_FORMAT_DEFINED * 196 BIN
ТАКОИ_ФОРМАТ_НЕ_ЗАДАН_ * 160 Q-GRAPH

(Глюк в тексте)
601124:        60287:         60300:
REF_0          GENERAL        SEQU__:
МЕТ.0          ГЛАВНЫЙ        СОРТ._:

60312:         60325:
BORDER_        _INTVL 002
_БОРДЮР        _ИНТЕРВ002

60358:         60387:
VERTICAL_LINE LINE_ACROSS
ВЕРТИК._ЛИНИЯ ГОРИЗ.ЛИНИЯ

60435:         60495:         60574:
BOX            TEXT           DATA_REF__:
ПР.            ТЕКС           ПОЛЕ-ДАН._:
```

(Примечание: перевод слова TEXT как ТЕКС - не очень удачный вариант, но мы ограничены местом в памяти. То же касается перевода слова BOX - ПР. (Прямоугольник). Как с этим бороться, мы ещё рассмотрим ниже.)

```
60591:          60603:
WIDTH_000       DEPTH_000
ШИРИНА000       ВЫСОТА000

60615:          60645:
NULL:           PAPER_
НОЛЬ:           БУМАГА

60677:
GIVE_REPORT_REF * AND
МЕТКА_ФОРМАТА__ * Q-GRAPH

60793:          60801:        60809:
INV_N           BRI_Y         PAD_Y
ИНВ.N           ЯРК.Y         ФОН_Y

60817:          60828:        60872:
FLASH_N         MPRT=42       LINE_000
МИГАН.N         СИМВ=42       _СТР.000

60883:          60931:
COL_000         X_COORD_000__LENGTH_000
КОЛ.000         Х-КООРД.000___ДЛИНА_000

60957:          61205:        61213:
Y_COORD_000     LENGTH=       WIDTH=
Y-КООРД.000     ДЛИНА__       ШИРИНА

61220:          61227:        61234:
DEPTH=          PAD           NULL_TEXT=
ВЫСОТА          ФОН           НОЛЬ_ТЕКСТ

61361:          61367:        61380:
LINE=           MICRO-PRT_    42_PITCH
_СТР.           МИКРОПЕЧ._    42_СИМВ.

61392:          61457:        61467:
COLUMN=         BRIGHT_       INVERSE_
КОЛОНКА         ЯРКОСТЬ       ИНВЕРСИЯ

61478:                        61484:
PAPER= * 189 ABS              FLASH_
БУМАГА * 225 LLIST            МИГАН.

61516:          61525:        61596:
X_COORD=        Y_COORD=      BORDER=
Х-КООРД.        Y-КООРД.      БОРДЮР_

61603:                        61612:
SEQUENCE= * 189 ABS           INTERVAL=
СОРТИРОВ. * 174 VAL$          ИНТЕРВАЛ_

61639:
AADD_NEW_FORMAT.RREVIEW_FORMAT..
АНОВЫЙ.ФОРМАТ...RИЗМЕНЕНИЕ_ФОРМ.

MMAIN_MENU......
МГЛАВНОЕ_МЕНЮ...

61688:
```

```
AADD_NEW_ELEMENTRREPLACE_ELEMENT
АНОВЫЙ_ЭЛЕМЕНТ..RИЗМЕНЕНИЕ_ЭЛЕМ.

EERASE_ELEMENT..NNEXT_ELEMENT...
ЕУДАЛЕНИЕ__ЭЛЕМ.NВЫБОР_ЭЛЕМЕНТА.

CCOPY_FORMAT....XDELETE_FORMAT..
СКОПИРОВ. ФОРМ..ХУДАЛЕНИЕ_ФОРМ..

DDISPLAY/PRINT..MPREVIOUS_MENU..
DДИСПЛЕИ/ПЕЧАТЬ.МПРЕДЫДУЩЕЕ_МЕНЮ

61817:
DDATA_FROM_RECD.LLITERAL_TEXT...
DДАННЫЕ.........LЗАГОЛОВОК,ТЕКСТ

BBOX............HHORIZONTAL_LINE
ВПРЯМОУГОЛЬНИК..НГОРИЗОНТ._ЛИНИЯ

VERTICAL_LINE.. MPREVIOUS_MENU..
VВЕРТИК._ЛИНИЯ..МПРЕДЫДУЩЕЕ_МЕНЮ

62102:          62127:
TOTAL____=_     AVERAGE_=_
ОБЩЕЕ____=_     СРЕДНЕЕ_=_

62194:          62219:
GIVE_FILE_NAME  ____ERROR__
ИМЯ_ФАИЛА______ __ОШИБКА__

63177:          63196:        63205:
REPORT_         MENU          .._MORE
ФОРМАТ_         МЕНЮ          .._ЕЩЁ_

63225:
f1  A  f0  LL_  f1  S  f0  INGLE_PAGE  * 197 OR
f1  A  f0  -ВСЕ,  f1   S  f0  -ЭКРАН__ * 160 Q-GRAPH
```

(здесь f1 и f0 - управляющие символы с кодами 18, 1 и 18, 0 - включение и выключение режима мигания)

```
63331:
NNEXT_PAGE......#ADVANCE_1-9_RCS
NСЛЕДУЮЩИЙ_ЭКРАН#ВПЕРЕД_НА_1-9..

OBACK_ONE RECORDBBACK_TO_1ST_REC
ОНАЗАД_НА_1.....ВВОЗВР._В_НАЧАЛО

PPRINT..........UUPDATE_TOP_REC.
РПЕЧАТЬ.........UИЗМЕН._ВЕРХ_ЗАП

EERASE_TOP_REC..OOMIT_TOP_RECORD
ЕУДАЛЕН.ВЕРХ_ЗАПОИСКЛЮЧ.ВЕРХ_ЗАП

CCOPY_TOP_RECORDSSEARCH_THE_FILE
СКОПИР. ВЕРХ_ЗАПSПОИСК..........

TTOTAL/AVERAGE..RSELECT_REPORT..
ТИТОГ_ОБЩ./СРЕДНRВЫБОР_ФОРМАТА..

MMAIN_MENU......QQUIT_THIS_MENU.
МГЛАВНОЕ_МЕНЮ...QУДАЛИТЬ_МЕНЮ...

64522:
ARG_NOT_NUMERIC  *  195 NOT
```

```
АРГ. НЕ_ЧИСЛОВОИ *  234 REM

64555:          64570:           64580:
ARGUMENT=       ALL_             SEL_
АРГУМЕНТ=       ВСЕ_             ВЫБР

64596:          64607:
CHAR_           NUM__
ТЕКС.           ЧИСЛ.

64623:
ASELECT_FROM_ALLLSELECT_FROM_SEL
АВЫБОР_ИЗ_ВСЕХ..LВЫБОР_ИЗ_ВЫБРАН

DDISPLAY/PRINT..MMAIN_MENU......
DДИСПЛЕЙ/ПЕЧАТЬ.MГЛАВНОЕ_МЕНЮ...

64688:
CCHARACTER......NNUMBERIC.......
CТЕКСТОВЫИ......NЧИСЛОВОИ.......

MPREVIOUS_MENU..NПРЕДЫДУЩЕЕ.МЕНЮ

64737:
EEQUAL_TO.......UUNEQUAL_TO.....
EРАВНО_АРГУМЕНТUНЕ_РАВНО_АРГ...

LLESS_THAN......GGREATER_THAN...
LМЕНЬШЕ.ЧЕМ_АРГ.GБОЛЬШЕ.ЧЕМ_АРГ.

SSTRING_SEARCH..MMAIH_MENU......
SСКАНИРОВАНИЕ...MГЛАВНОЕ_МЕНЮ...

65210:
NON-NUMERIC_DATA: f1  S  f0 KIP_
НЕ_ЧИСЛ.ДАНН. f1  S  f0 -ДАЛЬШЕ,

f1  U  f0  PDATE  * 197 OR
f1  U  f0  -СТОП  * 240 LIST

65320:
PPROGRAM+FILE...FFILE_ONLY......
PПРОГР.+_ФАЙЛ...FТОЛЬКО_ДАННЫЕ..

65353:
SAVE_NAME_?
ИМЯ_ФАЙЛА_?
```

После того, как перевод закончен на бумаге, загружаем программу-монитор. Убираем REM из строки 230 и начинаем замену английских текстов на русские в соответствии со своими записями. Не забывайте периодически выгружать результаты через RUN 5.

Попутно следует сказать о том, что не стоит экономить ленту, стирая старый вариант и записывая на него новый. Это позволит Вам вернуться назад в том случае, если в результате работы Вы случайно "запортите" программу, ошибочно изменив содержимое памяти где-нибудь в области машинных кодов.

Когда эта часть работы будет выполнена и записан последний вариант "русского" кодового куска, надо проверить, как все будет работать в новом виде. Запустите программу MF 09 и, после загрузки блоков, "MF LOADER" И "MF 09 LEER" остановите магнитофон и вставьте кассету с записью "русского" кодового куска. Загрузите его. После старта программы, если Вы все сделали аккуратно, Вы можете насладиться результатами своего труда. Но это потом. А сейчас надо внимательно посмотреть, нет ли где-нибудь

грамматических ошибок, которые практически неизбежны в результате такой большой работы, и вообще, все ли выполняется "ладно" и хорошо, нет ли где-нибудь "шероховатостей" перевода и т. д.

Если ошибок нет, то в общем можно считать работу выполненной, но присмотримся к работе программы повнимательнее. Ну например, вместо слова "BOX" - прямоугольник - мы ввели "ПР." (см. ячейку 60435). Это выглядит не очень удачно на экране, но, с другой стороны, что еще можно разместить в выделенных для этого трех байтах текста? Аналогично, вместо "TEXT" - мы ввели "ТЕКС" (ячейка 60495). Та же история - не хватает места. Что тут можно сделать?

Вспомним, как в машинных кодах осуществляется вывод на экран. Непосредственно перед выводом где-то должен быть задан адрес начала строки текста. Надо найти этот адрес и изменить его, расположив альтернативные текстовые сообщения на новом месте. Вот как это может выглядеть.

Текстовая строка "BOX" начинается фактически с адреса 60430:

```
60430   22       ?  - AT  2:0;
60431   2        ?
60432   0        ?
60433   17       ?  - PAPER  6;
60434   6        ?
60435   66       B  - т е к с т
60436   79       O
60437   88       X
60438   255      COPY     - "конец"
```

Вместо "ПР." желательно было бы разместить ну хотя бы:

```
AT 2 0 PAPER 6 ПРЯМОУГ. "конец"
```

Для такой строки надо 14 байтов памяти. Аналогично со строкой "TEXT" - фактическое начало - адрес 60490. Замена:

```
AT 2 0 PAPER 6 ТЕКСТ "конец"
```

Для этой строки надо 11 байтов памяти. Всего для двух строк нам нужны 25 байтов. Подумаем где бы их найти. В нашем случае есть свободное место перед символьным набором (расположенным с адреса 56560). Запустите программу-монитор (REM - в строке 321) и с адреса 56560-25=56535 осуществите ввод. Вначале пять пробелов (они нужны, чтобы потом на их место поставить управляющие коды AT и PAPER), затем текст "ПРЯМОУГ.", затем ещё пробел. И сразу же далее: еще пять пробелов, затем слово "ТЕКСТ" и еще один пробел. Ввод закончится ячейкой 56559. Теперь остановите программу, подставьте REM в строку 230 и удалите REM из строки 231. Запустите программу-монитор: GO TO 200 и снова с адреса 56535 введите следующие коды: 22, 2, 0, 17, 6, затем восемь раз просто нажмите "ENTER", ничего не вводя, затем введите: 255, 22, 2, 0, 17, 6, ещё 4 раза нажмите "ENTER", и, наконец, введите последнюю цифру 255. Ввод завершен на ячейке 56559.

Теперь надо найти в программе те места, где указаны адреса старых текстовых сообщений и изменить их на новые. Старый адрес для сообщения "BOX" - 60430. Для поиска подставим REM в строки 230 и 231 и введем новую строку:

```
210  IF (PEEK A+256*PEEK(A+1))<>60430 THEN GO TO 300
```

Запустите программу RUN, задайте адрес 56328 - это будет начало поиска - и можете откинуться на спинку кресла, пока компьютер не выдаст Вам необходимую информацию. На экране напечатается:

```
60405   14   ?
```

Не нажимайте "BREAK", подождите, пока не будет просмотрена вся программа, может быть на этот адрес есть ссылки и в других местах или это случайная комбинация чисел, никакого отношения не имеющая к процедуре вывода. Поиск закончится сообщением об ошибке: "B Integer out of range, 210:1". Это значит, что вся память просмотрена.

Место в программе, где указывается на искомый адрес, оказалось единственным. Тем проще для нас.

Теперь найдем адрес, где указано начало второго, интересующего нас сообщения - "TEXT" - это 60490. Изменим в строке 210 программы-монитора число 60430 на 60490 и GO TO 200. На экране появилось:

```
60458   74   J
```

Сообщение об ошибке. Поиск закончен, этот адрес тоже единственный. Теперь оба адреса надо заменить на новые. Для сообщения "ПРЯМОУГ. " это будет адрес 56535, а для "ТЕКСТ" - 56549. Младшие и старшие байты этих чисел находим, выполнив:

```
PRINT  56535-256*INT(56535/256)
PRINT  INT(65355/256)
PRINT  56549-INT(56549/256)
PRINT  INT(56549/256)
```

Получим, соответственно, числа 215, 220, 229, 220. Теперь выполним:

```
POKE 60405,215     POKE 60406,220
POKE 60458,229     POKE 60459,220
```

Осталось готовую программу записать на ленту, но прежде чем сделать RUN 5, вспомните, что наша программа стала длиннее на 25 байтов. Это надо учесть, производя изменения в строке 6 программы-монитора:

```
SAVE N$ CODE 56560,8976
```

надо заменить на

```
SAVE N$ CODE 56535,9001
```

Теперь можете сделать RUN 5. Надо также изменить программу-загрузчик "MF LOADER", заменив CLEAR 56559 на CLEAR 56534.

Попробуйте новый вариант программы. Вы согласны с тем, что она выиграла от такой замены?

Еще одна деталь. При запуске программы, когда появляется на экране главное меню, слева вверху на синем фоне появляется надпись

```
"FИЛE:MF 09 LEER"
```

Вместо русско-латинского слова "FИЛE" надо бы написать "ФАЙЛ", но вот беда: это слово мы не нашли в кодах программы MF 09. На самом деле все очень просто, Его там и нет. Надо искать в другом месте. Это слово находится внутри Бейсик-программы MF 09 LEER. Чтобы убедиться в этом, сделайте следующее. Остановите программу нажав в главном меню "L", затем "КУРСОР ВНИЗ" (CAPS SHIFT + 6). Теперь сделайте PRINT F$. Вы увидите надписи, выводимые на экран в режиме главного меню. Слово "FИЛE" находится в файле данных Бейсик-переменной F$. Как поступить в этом случае?

Запустите программу MF 09. Находясь в главном меню, нажмите "V", а затем "F" и, указав имя, например "LEER", запишите пустой файл на магнитофон. (Это как раз и будет переменная F$). Дальше загрузите опять программу-монитор (и коды MF 09), остановите ее и загрузите переменную F$, подав прямую команду:

```
LOAD "LEER" DATA F$ ()
```

Теперь надо помнить о том, что нельзя подавать команду RUN только GO TO 1, так как по команде RUN Вы уничтожите переменную F$. Просматривая дамп памяти примерно с адреса 24400, вскоре Вы встретите фразы: "MASTERFILE BEP 09" и "ФИЛЕ:LEER". Замените их на "MASTERFILE RUS 09" и "ФАЙЛ:LEER" обычным способом. Теперь остановите программу-монитор и запишите переменную F$ на магнитофон прямой командой:

```
SAVE "LEER" DATA F$ ()
```

Далее надо запустить программу MF 09 и, находясь в главном меню нажать "L". Задайте имя "LEER" и загрузите с магнитофона изменённый вариант переменной F$. После выхода в главное меню Вы увидите, правильно ли Вы все сделали. Теперь можете записать измененную Бейсик-программу MF 09, нажимая в главном меню "V" затем "P" и указав имя "MF 09 LEER".

Смотрим, что еще в программе не так. В режиме "дисплей" при просмотре записей в правом нижнем углу экрана на голубом фоне появляется надпись:

```
.._ЕЩЕ_ или No_ЕЩЕ_ ("_" - обозначает пробел)
```

При этом в памяти постоянно хранится только запись "_ЕЩЕ_", а "No" или ".." подставляются программой непосредственно в строку, подлежащую выводу, в зависимости от алгоритма работы. Таким образом, если мы хотим сделать перевод текста, то на место английского "No" не помещается даже русское "НЕТ". А русско-латинская фраза "No ЕЩЕ" выглядит довольно-таки неудачно. В этом случае можно попробовать использовать символы, например "+" и "-".

Анализируя работу программы MF 09 (кстати в этом мне очень помог трехтомник "Программирование в машинных кодах", написанный "ИНФОРКОМОМ"), я нашел, как мне кажется, приемлемый вариант замены. Для реализации этого варианта надо сделать:

```
POKE 62759, 230  POKE 63155, 32
POKE 63037, 230  POKE 63205, 32
POKE 63123, 230  POKE 62206, 32
POKE 63146, 230  POKE 62757, 45
POKE 63121, 43   POKE 63040, 45
```

Теперь, если при просмотре записей в режиме "дисплей" есть еще записи, то в левом нижнем углу будет: "_+_ЕЩЕ_", а если больше нет записей, то: "_-_ЕЩЕ_". По моему, это выглядят более гармонично.

Теперь еще об одной существенной детали. При вводе данных, для переноса строки в режиме "дисплей", применяется символ "вертикальная линия" - С.В.Л. (это в режиме "EXT. MODE" нажимаем SYMBOL SHIFT + S). Код этого символа равен 124. Но в кодах ASCII в символьном наборе КОИ-7 "НС", который применен в нашем варианте MF 09, коду 124 соответствует буква "Э". Поэтому, как только в тексте встретится буква "Э", она напечатана не будет, а произойдет перенос текста на новую строку.

Устранить можно и эту проблему, заменив символ, дающий команду переноса на какой-нибудь другой. Вопрос только: на какой? Ведь любой другой символ может встретиться в файлах данных, введенных на нерусифицированной программе MF 09.

Я считаю, что мне удалось найти удачное решение. Сделайте:

```
POKE 58237,0
POKE 63812,0
```

Теперь вместо С.В.Л. для переноса строки будет использоваться любой символ, с кодом больше 127. Это любое ключевое слово, например "STOP" или "AT" или любое другое, которое Вам удобнее набирать. При этом в файл данных будет занесен код 0 (независимо от того, какое ключевое слово Вы ввели). Символ с кодом 0 - это в кодах ASCII управляющий символ "NUL", он для печати не используется. Но в режиме "дисплей" теперь по этому

символу произойдет перенос на новую строку. Если же Вы при помощи переделанной программы будете просматривать файлы данных, введенные на непеределанной программе, то все символы переноса строки, находящиеся в тексте, станут видимыми.

Если при пользовании русифицированной программой MF 09 возникнут какие-либо неудобства, связанные с переносом строки, то напишите в ИНФОРКОМ, я готов придумать что можно сделать. (Например, можно написать несложную программу обработки уже готовых файлов данных, заменяющую все встречающиеся в тексте С.В.Л. на управляющие символы "NUL".)

Многие проблемы, существенные и несущественные, связанные с русификацией MF 09, мы решили. Но осталась еще одна очень важная проблема, пока еще нерешенная. Это сортировка по русскому алфавиту. Как производится сортировка по латинскому алфавиту? Очень просто. В кодах ASCII латинские буквы уже расположены в алфавитном порядке. Значит, располагая записи по возрастанию кодов, можно производить сортировку. С точки зрения программирования это довольно простая задача. С русским алфавитом сложнее. Здесь нет никакой связи между алфавитом и кодом символа. Поэтому сортировка по тому же принципу, что и латинского текста не годится. Причем проблема сортировки по русскому алфавиту гораздо шире, чем рамки программы MF 09. По-моему проблема вполне достойна того, чтобы вынести ее на общественный "Форум". Может быть, кому-нибудь из читателей удастся преодолеть ее? В таком случае заявите в ИНФОРКОМ. В свою очередь, я тоже пытаюсь найти приемлемый вариант решения этой задачи и когда закончу работу, сообщу об этом.

ЗАКЛЮЧЕНИЕ.

Мы закончили перевод программы MF 09 на русский язык. Выполняя эту работу, Вы познакомились с некоторыми методами, которые могут применяться для этого и немного "набили руку". Теперь Вы легко сможете применять "русский язык" при разработке своих программ.

Что же касается перевода фирменных программ, то тут следует отметить некоторые моменты. Приобретенные знания помогут Вам во многих случаях, однако будут встречаться разные "сюрпризы и хитрости", придуманные авторами (как было в MF 09, например, код последнего символа на 128 больше и др.). Процедуры вывода на экран могут быть самыми различными. Кроме того, трудности начнутся с загрузки программы, так как при нажатии "BREAK" программа "зависает" или происходит рестарт компьютера. Но даже если Вы удачно "взломаете" защиту загрузчиков программы, надо учесть, что очень многие коммерческие программы закодированы и поэтому, просматривая дамп памяти, мы ничего похожего на текстовые сообщения увидеть не можем. В этом случае надо, дизассемблируя программу, начиная со стартового адреса, найти процедуру декодирования, написать аналогичную процедуру в машинных кодах, раскодировать программу и только после этого заниматься переводом. Закодировать программу после перевода нет необходимости, проще изменить процедуру запуска, исключив элементы декодирования. Без знания машинных кодов Z-80 решить эту проблему, конечно невозможно.

*   *   *



"ИНФОРКОМ" получает массу писем от начинающих программистов с просьбами посоветовать им над чем бы стоило поработать, в какой области приложить свои усилия.

Мы считаем, что лучше всего в этой деле равняться на классиков. Поэтому пользуясь тем, что этот номер "ZX-РЕВЮ" является апрельским, мы поздравляем дорогих читателей с праздником 1-го апреля и рекомендуем обратить внимание на последние достижения всемирно известной фирмы PFA.

Мастеров "самого быстрого ксерокса" просим при перепечатке (или ином воспроизведении информации) не стесняться и смело указывать на первоисточник.

Проблем совместимости больше не существует.

Уникальная развлекательная программа разработана и выпущена фирмой PRESENT FROM AFRICA. Это первая версия имитатора "Super Real Bomb Simulator".

Программа отличается от всех известных аналогов тем, что ее не надо инициализировать, настраивать и вообще ее не надо загружать. Она не требует ни джойстика, ни клавиатуры.

Вам достаточно вставить кассету в магнитофон, нажать клавишу PLAY и спокойно ждать, когда она сработает.

Фирма особенно удовлетворена тем, что до сих пор от зарегистрированных пользователей не поступило ни одной жалобы (письма и заявления от родственников и друзей покойного фирма традиционно не рассматривает).

За особо дополнительную плату программа может поставляться на диске.

Решается вопрос о перспективах маркетинга программы в термоядерном исполнении (так называемый экспортный вариант). Это первая в мире программа, работающая на компьютере любой системы!

Проблем с продуктами больше нет!

Впервые в мире фирмой PRESENT FROM AFRICA выпущен имитатор продукта питания. Это завоевавшая всемирную известность программа "Shashlyik Basturma Simulator".

Программа состоит из двух частей. Первая часть - аркадная игра по нанизыванию кусочков баранины и лука на палочку.

Вторая часть требует специального блока питания (поставляется вместе с программой).

Подключив его и запустив программу, Вы уже через несколько минут сможете насладиться ароматом настоящего шашлыка, исходящим от Вашего компьютера.

Если запах немножко не тот, значит Вы забыли замочить компьютер в слабом растворе уксуса или в сухом вине. Сделать это лучше всего накануне вечером. Не забудьте добавить лука репчатого, перца молотого и посолить.

Суперновинка!

Развивая плодотворную идею программы "Shashlyik..." фирма PFA подготовила совершенно оригинальный программный продукт, объединяющий в себе абсолютно все известные жанры игровых, прикладных и вкусовых программ. Эта новинка поступила на прилавки под названием "Chuckie Egg F16".

Сначала Вы работаете в режиме "TOOLKIT", конструируя себе цыпленка-табака по вкусу. В этом режиме программа является также обучавшей по анатомии пернатых.

Увлекательный аркадный эффект возникает, если Вы ошибетесь и Ваш цыпленок начнет походить на утку или гуся - Вам придется отбиваться от группы сумасшедших яблок (APPLES), желающих в нем расположиться.

Вы можете выбрать своему цыпленку лапки, ножки, крылышки, грудку по вкусу, можете оснастить его хвостовым, оперением, подвесить дополнительные топливные баки, ракеты "воздух-воздух" и "воздух-земля", установить приборы ночного бомбометания и многое, многое другое.

Дальше программа может развиваться как STRATEGY/ACTION, а можно сразу перейти в кулинарный модуль с помощью описанного выше блока питания. Но в последнем случае способ предпусковой подготовки компьютера путем замачивания будет несколько отличаться, впрочем вместо инструкции к программе прилагается книга о вкусной и здоровой пище.

Проблем с ошибками больше нет!

Известно сколько неприятностей доставляют каждому работающему с компьютером сообщения об ошибках.

Наконец-то эта проблема решена всемирно известной фирмой PRESENT FROM AFRICA.

Покупайте новую программу "O.K. Simulator" (Имитатор O.K.)

Что бы Вы ни делали с компьютером, он всегда выдает радостное сообщение O.K.

Программа проста в работе. С ней успешно (и без ошибок) работают грудные дети. Группа академиков освоила работу с этой программой всего за полтора месяца во время учебного круиза по маршруту Одесса-Афины-Кейптаун-Сингапур-Токио и обратно. Теперь у них тоже все O.K.

По окончании круиза все академики высказали два пожелания и поставили один проблемный вопрос.

Первое пожелание - сократить количество утомительных нажатий клавиш хотя бы до двух (большее количество мешает круизу).

Второе пожелание - повторить круиз для окончательного решения вопроса о рекомендации программы ко всеобщему внедрению.

Проблемный вопрос: почему во всех городах мы побывали по два раза, а в Токио только один?

Сейчас фирма работает над новой версией, печатающей сообщение "O.K." в любых необходимых количествах автоматически, но, как сообщают, работа над проектом может затянуться из-за неразрешимости проблемного вопроса. Он стал известен в академических кругах под названием "Токийский синдром".

Ряд крупных академических институтов уже взялись за его отработку, по крайней мере на моделях. В модельных экспериментах функции Токио выполняли Лондон, Париж и Нью-Йорк, но проблема еще далека от разрешения. На очереди Сан-Франциско и Рио-де-Жанейро.

Происшествия!

В московском отделении фирмы PRESENT FROM AFRICA совершено крупное хищение. Украдены персональный компьютер, монитор, курсор и джойстик.

При обыске у подозреваемого обнаружены компьютер, джойстик и монитор.

Фирма гарантирует всем гражданам, которым известно что-либо о местоположении курсора, призовое вознаграждение своим программным обеспечением.

Без курсора фирма не может продолжать работы над замечательными проектами.

СОВЕТЫ ЭКСПЕРТОВ

PROFESSIONAL TENNIS

"Dinamic" 1990 г.



Эксперт М.Аксюта
г. Днепропетровск.

Эта игра относится к имитаторам спортивных игр. Вам предоставляется возможность хорошо потренироваться и сыграть на Уимблдонском турнире со звездами мирового тенниса. Но начнем по порядку. После загрузки игры перед Вами основное меню:

0. TORNEO (турнир)
1. CONTROLES (органы управления)
2. EQUIPAMIENTO (экипировка)
3. ENRENAMIENTO (тренировка)
4. CARACTERISTICAS (условия игры)

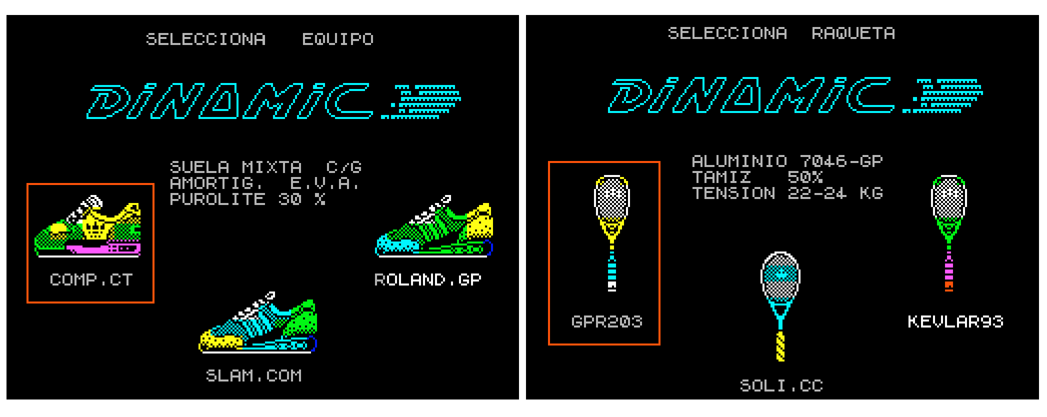


1. Controles

В этом режиме Вы можете ввести раскладку клавиатуры для управления двумя игроками: TECLADO_A и TECLADO_B

2. Equipamiento

Здесь Вы можете выбрать себе экипировку, а именно: - сначала кроссовки, а затем ракетку.



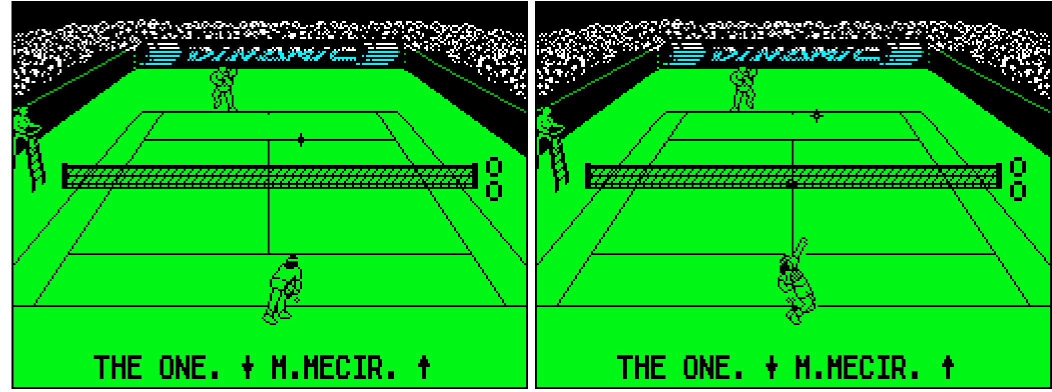
3. Entrenamiento

Вам предлагается потренироваться перед игрой на Уимблдоне. Вы можете выбрать тренировочный режим из следующего субменю:

1. PARTIDO
2. SAQVE
3. VOLEA

Выход из режима тренировки - нажатием клавиши "R".

Saqve - отработка подачи. Выбираете себе управляющий орган (например Кемпстон-джойстик) и проводите тренировку.



Нажмите кнопку "огонь" и переместите появившуюся на экране мишень в то место площадки, куда бы Вы хотели послать мяч. Подавать мяч нужно обязательно в накрест лежащий квадрат поля противника. Так, если Вы находитесь в правом нижнем квадрате, то подавать можно только в левый верхний. При ошибке или попадании мяча вне разрешенной зоны загорается сообщение и Вам дается вторая попытка, как в настоящей игре. Отработав подачу, можно нажатием "R" вернуться в субменю и перейти к отработке приема мяча.

Volea - отработка приема мяча. Ваш игрок может свободно перемещаться по площадке. Удар он выполняет нажатием кнопки "огонь". Если одновременно нажать "огонь" и "вверх", то удар получается более сильным, но пользоваться этим следует с осторожностью - когда Вы находитесь на задней линии или противник вышел близко к сетке, иначе мяч после Вашего удара может улететь за пределы площадки.

Если совместить удар с движением влево или вправо, то можно соответственно послать мяч влево или вправо, но это происходит не всегда и зависит от того, как Вы стоите относительно мяча. Так, если Вы стоите слева от мяча, он может полететь либо влево, либо прямо и наоборот, если Вы стоите справа от мяча во время приема.

Потренировавшись в приеме подачи и закончив тренировку клавишей "R", Вы можете сыграть пробную игру с компьютером, нажав клавишу "1" - Partido и выбрав для него силу игры "ORDENADOR".

Если Вас при этом не устраивают кроссовки или ракетка, вернитесь в главное меню и поменяйте их.

4. Caracteristicas

Вам предлагается следующее меню:

1. TIPO DE PISTA....
2. CAMBIO DE CAMPO....
3. NUMERO DE SET....

Tipo de pista - выбор покрытия корта, на котором Вы будете тренироваться. Вам будут предложены следующие варианты:

- TIERRA - грунтовое,
- HIERBA - травяное,
- RAPIDA - пластиковое.

Cambio de campo - определяет будет ли производиться смена сторон поля по ходу игры.

Если Вы хотите играть все время с одной стороны, выбирайте NO (НЕТ), если же

желаете проводить смены полей, выберите SI (ДА). Начинающим рекомендуется выбирать NO.

Numero de set - количество сетов (1-3-5), которое будет разыграно с компьютером при тренировке.

0. Torneo

Здесь разыгрывается турнирная игра. Сначала Вам предложат выбрать количество участников: 1-4 и ввести их имена, после чего Вы попадете в следующее субменю:

0 - CLASSIFICATION ATP

1 - INICIAR TEMPORADA

2 - CONTINUAR TORNEO

3 - TABLA DEL TORNEO

Classification ATP – таблица участников турнира, их номера. Вы всегда начинаете с самой низшей ступени.

Iniciar temporada - ввод имени игрока.

Continuar torneo - выбор управляющего органа и начало (продолжение) игры. Игра может быть прервана для перенастройки нажатием клавиши "R".

Tabla del torneo - таблица пар участников.

Итак, предварительно потренировавшись и выиграв у компьютера 3-4 сета, Вы можете принять участие в настоящем Уимблдонском турнире. Разве Вам не хочется победить Беккера или Лэндла?!

SNOOKER

Эксперт Ескевич А.А. г. Новосибирск.

"Снукер" - разновидность биллиарда.

Ныне эта игра - одна из популярнейших не только на туманном Альбионе, но и в более, чем тридцати других государствах. В Великобритании действует огромная сеть клубов, объединяющая более чем 6 млн. любителей снукера - это при том, что все население страны - 54 миллиона человек!

На биллиарде британцы играют уже свыше 100 лет. Таков же возраст и снукера. Происхождение этой игры точно не установлено. По одним сведениям снукер был придуман в Индии, а затем завезен в Англию, где и получил признание и широкое распространение. Другие знатоки утверждает, что в 1875 году желание как-то разнообразить обычный биллиард толкнуло одного из младших офицеров британской армии на то, чтобы включить в игру шары другого цвета.

Что же касается самого слова "снукер", то оно произошло от английского "snook", что в переводе означает "длинный нос". Снукерами же в британской армии называли юных кадетов только что начавших службу.

Так что же такое снукер?

Точный удар, уверенная срезка - и по зеленому сукну с мягким стуком раскатилось 22 разноцветных шара, останавливаясь в неожиданных положениях...

Это снукер - почти неизвестная в нашей стране игра на биллиарде. Изобретенная во второй половине прошлого века, игра оказалась сложнее, чем известные "Американка" и "Русская пирамида". Она скорее пробуждает интерес к сложным движениям шаров, чем к выигрышу.

Правила игры.

В снукер обычно играют вдвоем, но могут играть и несколько игроков.

Принцип очередности удара простой: если удар не принес очков, бьет следующий игрок.

Цветные шары оцениваются в зависимости от цвета:

- 15 красных шаров - по 1 очку;

- один желтый - 2 очка;
- один зеленый - 3 очка;
- один коричневый - 4;
- один синий - 5;
- один розовый - 6;
- один черный - 7 очков.

Белый шар - это биток. Только им можно бить по остальным шарам.

Разбивающий пирамиду может установить биток в любом месте зоны "дома" (площадь, ограниченная полукругом) - там, откуда удобнее бить.

Исходная расстановка шаров показана на рисунке. Вместо цвета шагов мы поставили их оценку в очках.

Первый ударом нужно сыграть только красный шар, а если биток заденет за какой-нибудь другой, игроку засчитывается ошибка и списываются очки в зависимости от ценности затронутого шара. Самый первым ударом важно не только удачно разбить пирамиду, но и отогнать биток как можно дальше от нее, создавая другому игроку позицию посложней.

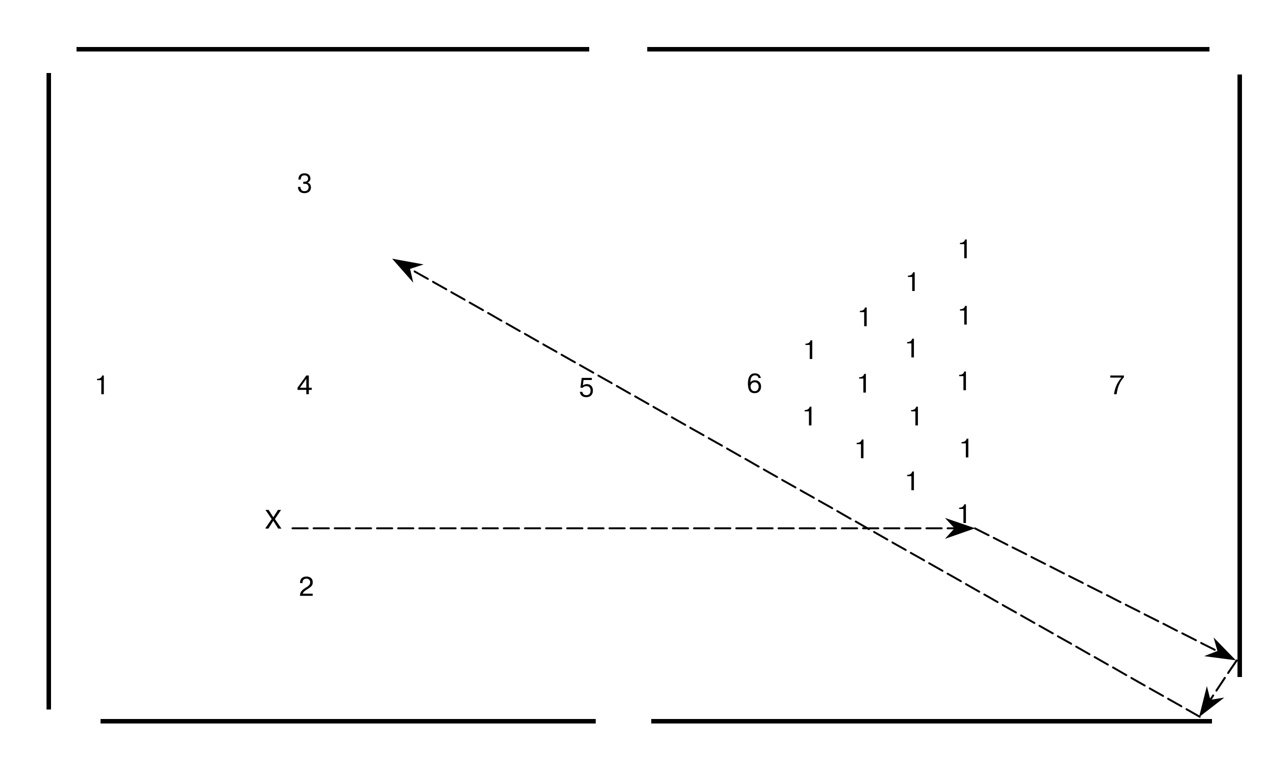
На рисунке классический начальный удар показан пунктиром.

Шары кладутся в любую из шести луз. За уложенный красный шар засчитывается одно очко. После этого биллиардист имеет право удара по любому цветному шару (цветными называются все шары, кроме красных и битка). Если начинающий игрок более уверен в каком-нибудь шаре, то он может бить и красный - один, другой, а потом, подогнав биток к цветному, уложить его в лузу.

Нужно твердо запомнить, что перед каждым цветным шаром должен быть забит красный!

Профессионалы снукера иногда усложняют игру тем, что после красного шара можно играть только цветной шар. Такая тактика всегда помогает набирать крупные серии. (Серия - сумма выигранных очков в течение одной очереди).

Например, вот какая серия может получиться, если было уложено в лузу несколько шаров: красный (1 очко), зеленый (3), снова красный (1 очко), розовый (6), красный (1), синий (5) и т.д.



Положенный в лузу красный шар выходит из игры. До тех пор, пока на столе есть хотя бы один красный шар, можно забивать цветные шары в любом порядке по своему выбору, но всякий раз перед цветным игрок должен уложить в лузу красный шар. Забитые цветные шары на этой стадии игры выставляются на те точки стола, где они стояли в начале игры.

Первая часть игры кончается, когда все красные шары забиты в лузы. Теперь правила по отношению к цветным шарам становятся более строгими: их надо класть в лузу в порядке их ценности, начиная с желтого (забив желтый, можно класть зеленый, затем коричневый, синий, розовый и черный. Уложенные таким образом цветные шары в игре уже больше не участвуют. Однако, если был сделан неправильный удар - биллиардист совершил ошибку при ударе, - цветной шар выставляется на свою основную позицию.

Игра подходит к завершению, когда на столе остается лишь 2 шара: "последний черный" и биток.

Бывает, что у партнеров примерно равное количество очков. Поэтому все решают эти семь очков за черный шар. Когда "последний черный" сыгран, или совершена ошибка - игра заканчивается. Редко, но может случиться и ничья - в этой случае для выявления победителя разыгрывается повторение "последнего черного". Черный шар выставляется на свое место, и по нему играют битком из "дома".

Когда знатоки биллиарда делают красивую игру, они не только укладывают шары в лузу, но при этом еще и маневрируют битком по всему полю. В результате маневра противнику может быть поставлен снукер - это кульминация игры, ее высший смысл, который и дал ей такое название. Снукер - это ловушка, из которой может выбраться только осторожный и хладнокровный игрок.

Снукер - это позиция, из которой биток не может достать тот шар, который по правилам должен быть сыгран: он либо "замазан", либо даже полностью закрыт, но шанс у Вас все-таки есть - выручит меткий бортовой удар.

А теперь ошибки, часто встречающиеся в снукере.

Случается, что игрок промахивается по шару, но это еще полбеды: этот шар, оттолкнувшись от борта, может задеть не тот, который должен быть сыгран. Если биток завалится в лузу, даже вслед за забитым шаром, эта ошибка наказывается штрафом в 4 очка. При назначении штрафа берется во внимание оценка шара, который следовало сыграть, или оценка шара, который сыгран неправильно. По правилам всегда берется наивысшее количество очков. Поэтому, если был правильно сыгран красный (или желтый, зеленый, коричневый) шар, но биток закатился в лузу, то списывается минимальное количество очков - 4; если же в таком положении окажется черный шар, штраф составит уже 7 очков (соответственно 6 - за розовый и 5 -за синий).

Когда нужно играть, например коричневый шар, и игрок делает правильный удар по нему, а закатывается какой-нибудь другой, то очки не списываются, а уложенный шар выставляется по правилам.

Подсчитано, что теоретический предел возможностей игрока в снукер - набрать за одну серию 155 очков (напомним, что серия - это сумма очков, выигранных за одну очередь).

Выдающийся мастер снукера англичанин Джо Дейвис установил в 1955 году рекорд мира, набрав серию в 147 очков. Может быть, Вы попробуете превзойти мастера?

Настройка программы.

После загрузки программа выдает следующее меню:

CONTROL OPTIONS	ОРГАН УПРАВЛЕНИЯ
1. KEYBOARD	1. КЛАВИАТУРА (O,P,Q,A,Enter)
2. KEMPSTON	2. КЕМПСТОН-ДЖОЙСТИК
3. CURSOR KEYS	3. КУРСОР (5,8,6,7,0)
4.INTERFACE 2	4. ИНТЕРФЕЙС 2 (Синклер-джойстик)
Press 5 to Continue	Нажмите "5" для продолжения.

Выбрав орган управления, нажмите "5" - появится следующее меню:

Game options:	Опции игры:
1. ONE PLAYER	- один игрок
2. TWO PLAYER	- два игрока
3. CURSOR SOUND ON	- звук курсора включен

4. CURSOR SOUND OFF - звук курсора выключен
5. LONG GAME - длинная игра
6. SHORT GAME - короткая игра
7.CURRAH SPEECH OFF - речь выключена
8. CURRAH SPEECH ON - речь включена
Press any other key to play — Для начала игры нажните любую другую клавишу.

Сделав выбор, нажмите клавишу, и на экране появится биллиардный стол с расставленными шарами.

Структура экрана.

В левом верхнем углу показаны набранные Вами очки (POINTS) и сумма штрафных очков за ошибки (FOULS), в правом верхнем углу - номер серии (VISITS).

Внизу показаны сила удара (POWER), биток для выбора направления вращения шара (SPIN) и то, какой шар нужно сыграть в данный момент (например, "RED WANTED" - "Требуется красный").

После того, как Вы забьете красный шар, появится надпись "COLOUR WANTED" ("требуется цветной"). Выберите цветной шар, который Вы хотите забить и нажмите клавишу, соответствующую стоимости этого шара (2-желтый, 3-зеленый, 4-коричневый, 5-синий, 6-розовый, 7-черный). Например, Вы решили сыграть черный шар - нажмите "7" и надпись "COLOUR WANTED" сменится на "BLACK WANTED". После этого Вы обязательно должны бить по черному шару. Если промахнетесь, или биток заденет сначала за какой-либо другой шар, Вам начислят штрафные очки за удар не но правилам.

Начало игры.

Перед началом игры Вы должны установить биток в любой точке "дома". Для этого переместите курсор в выбранное место и нажмите "огонь" (эту операцию Вам придется проделывать всякий раз после того, как биток угодит в лузу).

Теперь Вы должны нанести удар. Это несложно, но требует тщательной подготовки:

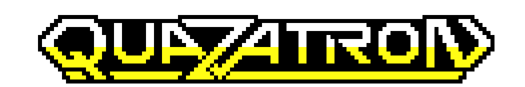
1. Установите курсор в то место, куда хотите направить биток и нажмите "огонь".

2. Установите силу удара клавишами влево/вправо, пользуясь шкалой внизу экрана, и вновь нажмите "огонь".

3. Установите курсор на ту точку битка, куда Вы хотите "ударить кием" (проще всего бить в центр битка, но если Вы научитесь "срезать" шар, это будет весьма полезно) и вновь нажмите "огонь" - удар произведен.

Теперь, когда правила известны, остается лишь приобрести навык - и победа будет за Вами! Дерзайте, будущие мастера международной игры на биллиарде!

QUAZATRON



Сегодня мы представим Вам программу QUAZATRON, разработанную Стивом Тернером, с работами которого наши читатели уже знакомы по циклу статей "Профессиональный подход".

В мае 1986 года известный журнал "Sinclair User" присвоил этой программе свой знак:

Sinclair User
CLASSICS

Этот красный с золотом знак присваивается только тем программам, которые не просто являются лучшими в своем жанре, а открывают новые жанры или новые направления в своем жанре или открывают новые, ранее неизвестные приемы и методы. Для программиста получение такого знака может быть и не столь значительно, как получение премии "Оскар" для кинорежиссера, но не менее дорого.

Конечно, сейчас уже далеко не 1986 год, но игра эта еще во многом свежа. Возможно, что в силу определенной сложности, она долгое время выпадала из поля зрения наших пользователей и об этом приходится только сожалеть. Ведь программа настолько многопланова и разнообразна, что возвращаться к ней можно много и много раз. Она представляет довольно гармоничный баланс между играми аркадного и стратегического жанра.

Квазатрон - это огромный подземный технополис, населенный роботами. Расположен он на планете Квартех. Несколько попыток земных экспедиций по проникновению в этот комплекс закончились неудачно. Вам предлагается попробовать свои силы и захватить город с помощью специального робота MECHNOTECH KLP2.

KLP 2 был сконструирован в качестве разведывательного робота и провел специальную подготовку для участия в исследовательских экспедициях в составе разведывательных команд. К сожалению, его пришлось отстранить от групповых операций в связи с его патологической страстью разбирать другие устройства (обратите внимание на то, что KLP 2 читается по-английски как KLPTWO - очень похоже на КЛЕПТО - и вспомните, что непреодолимое желание тащить к себе все, что плохо лежит, называется клептоманией).

Отправленный для рутинной патрульно-таможенной службы на границы галактики, КЛЕПТО, тем не менее, завоевал известность и стал общественным героем, когда утилизировал левую ногу знаменитого межзвездного пирата, робота-андроида человекоподобного типа.

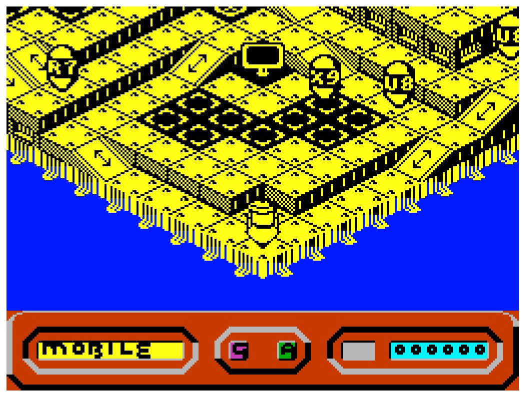
Вот почему о нем вспомнили, когда готовили миссию QUAZATRON.

Находясь под Вашим управлением, KLP 2 может работать в двух основных режимах:

- MOBILE - режим перемещения;
- GRAPPLE - режим захвата.

Когда он работает в режиме перемещения, игра развивается как обычная аркадная игра. Вы можете перемещаться по трехмерному уровню технополиса, можете периодически отстреливать враждебных роботов, можете перемещаться с уровня на уровень, можете подключаться к установленным в разных местах дисплеям или подпитываться энергией в специальных точках.

Подключение к дисплею позволяет войти в информационную сеть технополиса, через которую открывается доступ к самой разнообразной информации. Вы можете увидеть план уровня, схему расположения уровней в городе, можете получить справку о роботах-противниках, узнать их спецификацию.



Подзарядка энергией - тоже совершенно необходимое действие.

Как бы Вы ни работали, Ваш успех зависит от наличия энергии. Если ее мало, загорается надпись POWER, раздается неприятный звуковой сигнал и, если Вы сломя голову не броситесь к источнику энергии, можете считать, что миссия закончена. Самое интересное - это то, что для указания уровня энергии нет никаких счетчиков, никаких шкал, никаких указателей. Над головой KLP-2 есть небольшая "шапочка". Скорость, с которой она вращается, и зависит от уровня энергии.

Надо сказать, что обычное аркадное путешествие по этажам города закончится очень быстро. Дело в том, что среди населяющих его роботов есть такие, которые не дадут Вам ни малейшего шанса остаться в живых, если Вы попробуете перебежать им дорогу.

Вот тут-то аркадная игра и превращается в стратегическую (а если хотите, то в деловую, ведь менеджмент - тоже стратегия, хоть и отличается от военной). В этой фазе игры Вы работаете со своим роботом в режиме GRAPPLE. В этом режиме Вы можете войти в непосредственный контакт с противником, подключиться к его внутренним цепям и попытаться его перепрограммировать, если он не сделает то же с Вами. Здесь Ваш успех будет зависеть от быстроты реакции, от оснащенности Вашего робота и конечно от его энергетических возможностей.

Схватка в режиме GRAPPLE представляет собой аркадную вставку в основную игру и развивается на особом экране, но мы о ней писать не будем, предоставив чуть ниже слово нашему эксперту Курбацкому Андрею Александровичу из Казани, накопившему определенный опыт в этом деле.

Если Вам удастся захватить хотя бы семь блоков из двенадцати в логических цепях Вашего противника, то схватка считается закончившейся в Вашу пользу, робот противника переходит в подчиненное состояние и Вы можете приступить к его тестированию. Тестирование покажет, какие модули остались неповрежденными в результате схватки и Вы сможете принять решение о том, что Вам можно использовать для себя. Так, постепенно каннибализируя павших противников, Вы можете улучшать оснащенность своего робота.

Вот где открывается поле для стратегии.

Во-первых, Вы должны сами принять решение о том, с каким роботом Вам имеет смысл входить в тесный контакт, а какого достаточно просто расстрелять. Решение Вы принимаете, зная модель робота по его бортовому номеру, его спецификацию и состав его модулей.

Противник не может быть намного мощнее Вас - это шаг к могиле, но и не должен быть слишком слабым, т.к. если с него нечего взять, то и нет смысла расходовать на него время и силы, разве что для тренировки.



Во-вторых, сама схватка должна проходить с Вашей стороны достаточно интеллектуально, ибо после иного боя Вы можете уже не найти у противника необходимых Вам компонентов в исправном состоянии. Попробуйте в начале игры перестрелять мощных боевых роботов противника и Вам впоследствии уже негде будет взять особо ценные модули, необходимые для захвата всего города.

В-третьих, утилизация для своих целей исправных блоков от захваченных роботов тоже должна проходить с умом. Возьмите очень мощный двигатель - и с Вашим скромным энергетическим отсеком Вы устанете бегать на подзарядку, а если один раз не добежите - игре конец. Возьмите самое изощренное орудие и система защиты и на Вашем слабеньком шасси KLP-2 будет еле ползать по поверхности, а падение с небольшой высоты станет приводить к значительным потерям энергии. И так далее и тому подобное.

Таким образом, игра представляет такой сплав стратегических решений, что ее серьезный анализ вполне может занять Вас не на одну неделю и доставит огромное удовольствие.

В заключение этого общего обзора программы прежде, чем предоставить слово нашему эксперту из Казани, мы приведем небольшое интервью со Стивом Тернером, и заглянем краем глаза на ту кухню, где готовятся такие вкусные блюда для Вашего компьютера.

* * *

В: Скажите пожалуйста, как долго шла работа над программой QUAZATRON?

О: Эта работа затянулась более, чем на шесть месяцев. Я знаю, конечно, что многие заявляют о том, что вполне пристойная аркадная игра может быть сделана и за два месяца, но у меня особый случай - ведь и музыку и графику я тоже делаю сам. Я вообще все делаю сам, хотя концепцию программы и наброски широко обсуждаю с коллегами.

В: Вам удается обыграть собственную программу?

О: Вы знаете, после полугода работы с машинным кодом, я месяца два вообще не мог смотреть на эту программу. Потом постепенно успокоился и нередко играю в нее. Но ответить на Ваш вопрос я не могу, ведь я не играю на выигрыш. У меня совсем другая цель - я ищу способы, которыми можно было бы "завалить" программу.

В: Когда Вы поняли, что эта программа Вам удалась?

О: Только в самый последний момент. Здесь надо знать мой метод работы над программами. Я не вижу ее до конца работы, все процедуры отрабатываются по отдельности или группами, но окончательная сборка производится в самый последний момент. Конечно, многие вопросы, особенно связанные с графикой, приходится обкатывать на моделях, ведь здесь важно быстродействие процедур с одной стороны и художественное впечатление с другой, но это совсем не то же, что работа с программой.

В: Что Вам не удалось в программе?

О: Отсутствие гладкого скроллинга экрана. У меня, конечно, были идеи как его обеспечить, но неминуемо приходилось бы жертвовать чем-то другим и сейчас, оглядываясь назад, я полагаю, что так получилось даже лучше.

В: Ваша программа чем-то напоминает PARADROID, выпущенный незадолго до нее для "Коммодора-64"? Она была в какой-то степени для Вас источником вдохновения?

О: Нет, все гораздо проще. Просто мы с Эндрю Брейбруком, автором "Парадроида", работаем в одной комнате. Естественно, по ходу работы идет постоянный обмен идеями. Кроме того, очень часто совместно обсуждаются вопросы дизайна. А если уж говорить об источнике вдохновения, то им явилась в некоторой степени программа MARBLE MADNESS и, в какой-то степени, ALIEN-8.

В: И последний вопрос. В самом начале программы звучит удивительно сложная музыка, если вспомнить, что у "Спектрума" только один канал...?

О: Да, конечно, каждый знает, что звуковая система "Спектрума" совершенно не соответствует прочим его возможностям. Но мне удалось программным путем, основываясь на системе прерываний, подготовить пакет процедур, имитирующих многоканальный звук. В конце концов, все это свелось только к манипуляциям фазой и частотой.

* * *

Теперь мы рассмотрим практические вопросы, связанные с боевыми действиями на планете Квартех. Слово имеет Курбацкий А.А., г.Казань.

Все роботы, с которыми Вы можете встретиться в технополисе, имеют бортовую маркировку, с помощью которой можно получить более подробную информацию об этом роботе.

Маркировка состоит из буквы и цифры. Буквенная маркировка указывает на класс робота, а цифровая - на положение робота в этом классе.

Классы роботов:

A (AUTOMATION)
B (BATTLE)
C (COMMAND)
L (LOGIC)
O (MEDIC)
R (REPAIR)
S (SECURITY)
U (UTILITY)
X (MENIAL)

Не входят в эту классификацию только два типа роботов. Это AB (ANDREWOID) И ST (PROGRAMMER).

Цифровая кодировка (от 0 до 9) указывает на статус робота в своем классе. Чем она ниже, тем выше статус робота, тем более совершенными системами он вооружен, тем больше доступный для него запас энергии.

Все роботы классов AB, ST, A и S относятся к высшей касте и защищены наиболее сильно. К средней касте можно отнести роботов L, O, R. К низшей касте относятся роботы U и X и защищены они довольно-таки слабо и в цифровой нумерации не опускаются ниже 7.

Нажав клавишу "огонь", Вы переходите в режим GRAPPLE и, коснувшись любого робота, входите в бой. Бой за перехват управления выглядит так (см. рис. 1).

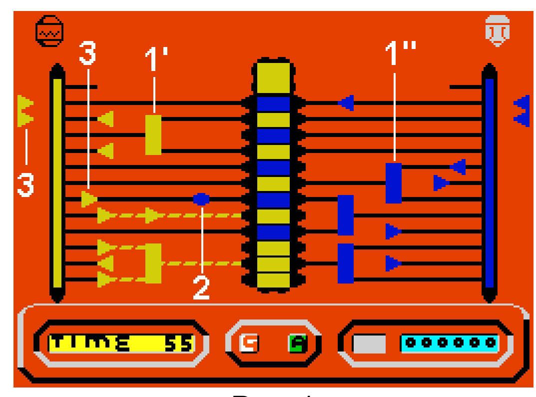


Рис.1

В начале игры у Вас всего три энергетические капсулы (3) которые Вы можете подсоединить к любому проводу, соединенному центральным блоком (см. рис.). Иногда на проводах стоят такие устройства как сумматоры (делители) (1), инверторы (2) и дополнительные энергетические капсулы (3). С помощью делителя (1') можно от одного провода запитать два и более отсека, сумматор же (1'') наоборот от несколько проводов запитывает один отсек. Инвертор (2) обращает Ваш сигнал и превращает в сигнал противника, так что Вам он мешает, а не помогает. Дополнительные энергетические капсулы могут помочь Вам увеличить время подпитки отсека, так как при израсходовании энергии в Вашей капсуле, подпитка блока прекращается.

Некоторые провода могут быть оборваны. Сбоку указано количество энергетических капсул, имеющихся в Вашем распоряжении.

Сначала Вам предоставляется право выбрать сторону, с какой Вы хотите вести бой. Это очень важный момент, от которого в значительной степени будет зависеть успех боя. После выбора стороны Вы можете с помощью клавиш "вверх", "вниз" подвести энергетическую капсулу к проводу, к которому Вы хотите подключиться, и нажать клавишу "огонь". Блок центрального отсека закрашивается. Конечная цель боя - захватить отсеков больше, чем Ваш противник. Всего на бой дается 60 секунд, желательно подключиться чуть позже, чем противник, так как это позволит при Вашем недостатке энергетических капсул продержаться как можно дольше и, в конечном итоге, захватить энергетический отсек. Если схватка закончится вничью, то Вам предоставляется другая энергетическая схема.

В случае победы противника, если это первый бой, Ваш робот разрушится, а если эта схватка происходила после того, как Вы уже кого-либо победили, то теряются все захваченные ранее блоки и оружие, а силовая установка остается почти без энергии. Тогда Вам после схватки нужно быстрее идти к энергетической клемме, расположенной на этаже, и подпитаться энергией. Маркировка последнего побежденного робота отображается в окне чуть правее центра пульта. Вот список устройств, которые Вы можете захватить, расположенный по степени приоритета.

Двигательная установка (DRIVE).
LINEAR MK 1
LINEAR MK 2
LINEAR MK 3
GRAVITRONIC MK 1
GRAVITRONIC MK 2
HEAVY DUTY
DUAL LINEAR
ULTRAGRAV

Оружие (WEAPONS)
PULSE LASER
DUAL LASER DESINTEGRATOR
AUTOCANNON

DISRUPTOR

Наибольшей разрушительной способностью обладает DISRUPTOR, который поражает всех роботов, видимых на экране. AUTOCANNON стоит на втором месте: пробивает любой корпус, но стреляет только в одном определенном направлении.

Имеет значительный вес. Все прочее оружие пробивает лишь определенные корпусы, да и то не всегда с первого раза.

Силовые установки (POWERS)
CHEMIFAX MK 1
ROBOTRONIC MK 1
ROBOTRONIC MK 2
ROBOTBONIC MK 3
TRIOBATIC
CYBONIC MK 1
CYBONIC MK 2

Чем выше класс силовой установки, тем более мощный двигатель и оружие в течение более долгого времени способна она поддерживать.

Шасси (CHASSIS)
DURALITE
TRIALIUM MK 1
TRIALIUM MK 2
CHROMITE
PLASTEEL
STRESSED PLASTEEL
CORALLOID MK 1
CORALLOID MK 2

Чем совершеннее шасси, тем легче Ваш робот перенесет падение с высоты.

Другие устройства (DEVICES).

Эти устройства являются дополнительными и потому они есть только у роботов высшего и среднего классов, у низших роботов они отсутствуют.

DISRUPTOR SHIELD - защита от воздействия DISRUPTOR-а. Она позволяет продержаться более долгое время при действии мощного оружия. Кстати, роботов, имеющих эту защиту, легче победить в борьбе за перехват управления, чем уничтожить оружием.

LASER SHIELD - защита от действия лазерного оружия, кстати неплохо защищает и от попыток пробить Ваш робот с помощью лобового тарана.

RAM THRUSTER - ускоритель. Позволяет увеличить скорость движения Вашего робота, способствует успеху тарана.

POWER BOOTS - обеспечивает уменьшение нагрузки на шасси за счет дополнительной тяги, что позволяет более стойко переносить падения с высоты.

DETECTOR - устройство, предназначенное для контроля за уровнем радиации. Сигнализирует о наличии опасного излучения.

OVERDRIVE - устройство для форсирования тяги двигательной установки.

Для тех, кто интересуется тем, какие устройства установлены на роботах, приводим список в таблице 1. Список не полный в связи с тем, что пока еще не все обнаружено. Желаем удачи в игре.

обозначение	класс	двигатель	силовая установка	оружие	шасси	дополнительное устройство
AB	альфа	ULTRAGRAV	CYBONIC mk 2	AUTOCANNON	CORRLOID mk 2	RAM TRUSTER
ST	бета	DUAL LINEAR	CYBONIC mk 1	DESINTEGRATOR	CORRLOID mk 1	DISRUPTOR SHIELD
		AUTOMATION				
A1	альфа	ULTRAGRAV	CYBONIC mk 2	AUTOCANNON	CORRLOID mk 2	LASER SHIELD
		BATTLE				
B2	бета	HEAVY DUTY	TRIOBATIC	DISRUPTOR	CORRLOID mk 1	DISRUPTOR SHIELD
B3	гамма	GRAVITRONIC mk 2	TRIOBATIC	DISRUPTOR	STR.PLASTEEL	DISRUPTOR SHIELD
B4	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 3	DISRUPTOR	STR.PLASTEEL	LASER SHIELD
B5	дельта	GRAVITRONIC mk 1	ROBOTRONIC mk 2	AUTOCANNON	STR.PLASTEEL	RAM TRUSTER
B6	дельта	GRAVITRONIC mk 1	ROBOTRONIC mk 2	DESINTEGRATOR	CORRLOID mk 2	LASER SHIELD
B7	дельта	GRAVITRONIC mk 1	ROBOTRONIC mk 2	DESINTEGRATOR	CORRLOID mk 1	RAM TRUSTER
		LOGIC				
L3	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 2	DUAL LASER	PLASTEEL	POWER BOOTS
L4	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 2	PULSE LASER	PLASTEEL	
L5	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 1	PULSE LASER	PLASTEEL	
L6	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 1	PULSE LASER	PLASTEEL	
		MEDIC				
00	гамма	GRAVITRONIC mk 2	ROBOTRONIC mk 1	PULSE LASER	PLASTEEL	POWER BOOTS
		REPAIR				
R5	дельта	LINEAR mk 3	CHEMIFAX mk 2	DUAL LASER	PLASTEEL	DISRUPTOR SHIELD
R6	дельта	LINEAR mk 3	CHEMIFAX mk 2	DUAL LASER	PLASTEEL	
R7	дельта	LINEAR mk 3	CHEMIFAX mk 2	DUAL LASER	TRIALIUM mk 2	
R8	дельта	LINEAR mk 3	CHEMIFAX mk 2	PULSE LASER	TRIALIUM mk 2	
		COMMAND				
C1	альфа	ULTRAGRAV	CYBONIC mk 1	DESINTEGRATOR	CORRALOID mk 1	POWER BOOTS
C2	альфа	DUAL LINEAR	CYBONIC mk 1	DESINTEGRATOR	CORRALOID mk 1	DETECTOR
C3	бета	DUAL LINEAR	CYBONIC mk 1	DESINTEGRATOR	CORRALOID mk 1	OVERDRIVE
C4	бета	DUAL LINEAR	CYBONIC mk 1	DESINTEGRATOR	STR.PLASTEEL	LASER SHIELD
C5	бета	HEAVY DUTY	TRIOBATIC	DESINTEGRATOR	STR.PLASTEEL	DISTRUPTOR
		SECURITY				
S2	бета	HEAVY DUTY	TRIOBATIC	AUTOCANNON	STR.PLASTEEL	DETECTOR
S3	бета	DUAL LINEAR	TRIOBATIC	AUTOCANNON	STR.PLASTEEL	DETECTOR
S4	бета	HEAVY DUTY	TRIOBATIC	DESINTEGRATOR	STR.PLASTEEL	DETECTOR
S5	бета	HEAVY DUTY	ROBOTRONIC mk 3	AUTOCANNON	STR.PLASTEEL	DETECTOR
S6	бета	GRAVITRONIC mk 2	TRIOBATIC	AUTOCANNON	STR.PLASTEEL	DETECTOR
		UTILITY				
U7	епсилон	LINEAR mk 2	CHEMIFAX mk 1	PULSE LASER	TRIALIUM mk 2	
U8	епсилон	LINEAR mk 1	CHEMIFAX mk 1	PULSE LASER	TRIALIUM mk 2	
U9	епсилон	LINEAR mk 1	CHEMIFAX mk 1		TRIALIUM mk 1	
		MENTAL				
X8	епсилон	GRAVITRONIC mk 2	CHEMIFAX mk 1		TRIALIUM mk 1	
X9	епсилон	LINEAR mk 1	CHEMIFAX mk 2		DURALITE	

CAPTAIN FIZZ



"Psyclapse" 1989 г.
Эксперт Троекуров В.И.
г. Киев

Программа "CAPTAIN FIZZ" принадлежит к аркадному жанру. Действие происходит на космическом корабле "ИКАРУС", который потерял управление и движется к Солнцу. Если он подойдет к нему слишком близко, то взорвется, разрывая галактику. Чтобы это предотвратить, необходимо телепортироваться на борт звездолета и разрушить компьютеры, управляющие его движением.

Палубы корабля представляют собой достаточно сложную систему лабиринтов и Ваша главная цель разобраться в этой системе, исследовать корабль и восстановить его управление.

Характерной особенностью игры является то, что она позволяет работать над главной задачей "кооперативно". Два игрока могут играть одновременно и помогать друг другу своими действиями.

Экран программы.

Экран разделен для двух игроков по горизонтали. Панель каждого игрока состоит из трех частей:

1. Информационное табло. Сюда выводится текущий результат игры (очки), шкала, показывающая температуру Вашего оружия (когда шкала исчезла от продолжительной стрельбы, необходимо отпустить кнопку и подождать, пока она восстановится).

ARMOUR - Ваша защита.

DAMAGE - Величина повреждений.

CHARGE - Заряд.

CREDIT - Количество "попыток".

2. Вторая часть - основной экран, на котором происходит действие игры. У каждого из игроков свой отдельный экран, что способствует независимости одного игрока от другого. Сверху над экраном расположены два слова:

HEALTH - состояние Вашего здоровья в 9999 энергетических единицах.

CARDS - карточки (пропуска), количество пропусков, их цвета (для каждого пропуска своя дверь, которую он может открыть).

3. Третья часть - индикатор состояния. Ломаная линия на нем - Ваша кардиограмма. По мере ухудшения Вашего самочувствия линия выпрямляется.

Здесь же представлены три дополнительных индикатора.

а) EXIT - "выход". Когда Вы ликвидировали все компьютеры на определенном уровне, то увидите, что загорится этот индикатор - значит можно телепортироваться на другой уровень. Открытый выход имеет овальную форму черного цвета с желтой окантовкой по краям. Закрытый выход имеет ту же форму но сверху на нем решетка желтого цвета.

б) SWITCHES - "переключатели". Здесь расположены четыре индикатора. Когда индикаторы горят, силовое поле отключено.

в) MINES - "мины". Индикатор предупредит Вас, что в помещении имеется мина и

лучше здесь не стрелять, т.к. при попадании в мину Вы потеряете при ее взрыве определенную часть энергии.

Программа имеет еще одну интересную особенность. После загрузки ее можно настроить на работу с одним из трех европейских языков:

1. Английский.
2. Немецкий.
3. Французский.

После ввода программы, нажмите клавишу "5" на клавиатуре (2 игрок) и "FIRE" на джойстике (1 игрок) и начинайте игру.

Итак, начинаем игру вдвоем. На экранах появятся два героя - один желтого цвета, другой - белого. Первый управляется от клавиатуры клавишами:

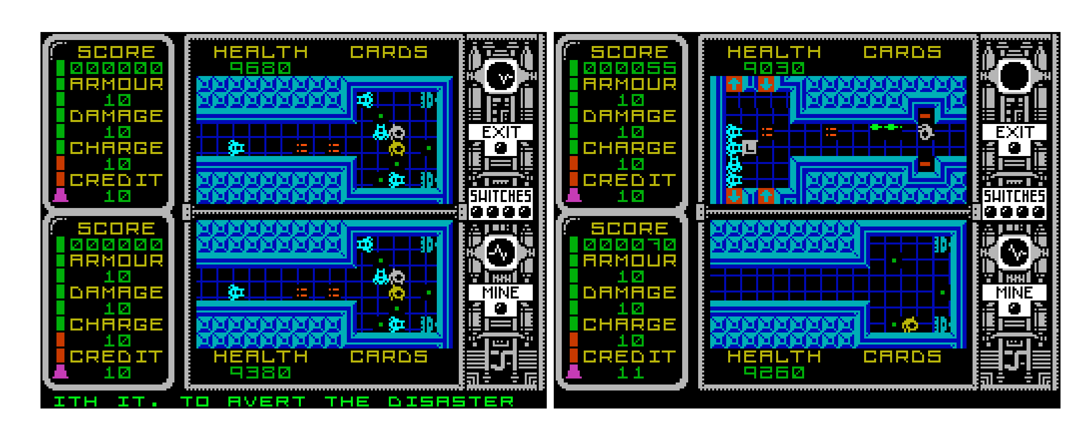
1 – влево 2 – вправо
3 – вниз 4 - вверх
5 - "огонь"

Второй герой может управляться как от джойстика, так и от клавиатуры:

6 – влево 7 – вправо
8 – вниз 9 - вверх
0 - "огонь"

Первое, что Вы должны сделать - это расстрелять компьютеры (они выглядят, как белые квадратики, маркированные буквой "L").

Сложность заключается в том, что их охраняют боевые роботы, с которыми предстоит провести перестрелку. Кроме того, есть небольшое силовое поле, способное какое-то время защищать компьютеры от поражения Вашим оружием.



Оба космонавта имеют по мощному бластеру с неограниченным боекомплектом. Но Вам следует внимательно следить за температурой своего оружия, так как оно не допускает перегрева и перестает стрелять. Кроме бластеров у космонавтов еще есть и супербомба, способная уничтожать враждебную боевую технику в пределах данного экрана, но не повреждающая компьютеры.

В исходном состоянии Ваша сила составляет 9999 энергетических единиц. Но здоровье можно поправить, когда Вы соберете энергетические аптечки (выглядят, как зеленые точки), но использовать их для своих целей можно только лишь подойдя к энергетической емкости. Чтобы понять, как выглядят энергетические емкости, надо обратить внимание на первую картинку игры. Рядом с двумя героями изображены зеленые точки (энергетические аптечки), а справа энергетические емкости.

Возьмите аптечку и подойдите к энергетической емкости. Вы увидите, что Ваша энергия возрастет до 9999 единиц - это предел и не тратьте лишние аптечки зря.

Карточки-пропуска понадобятся Вам, чтобы пройти в другие отсеки корабля. Пропуска выглядят как маленькие цветные прямоугольники. Они бывают 5 разных цветов: красного, желтого, синего, пурпурного и голубого. На дверях указан цвет пропуска, которым они открываются. Если Ваш пропуск не соответствует цвету на двери, то Вы не сможете ее открыть.

Есть двери, вырабатывающие защитное силовое поле, контактов с которым

необходимо избегать. В случае Вашей гибели задача партнера очень усложнится. Впрочем, если ему удастся пройти на следующий уровень, Вы появитесь снова, хотя уровень Вашей энергии при этом будет равен только 2000 единиц.

Гибель обоих игроков ведет к прекращению игры. После этого программа предложит Вам продолжить игру (CONTINUE). Утвердительный ответ (Y) позволяет вернуться на один-два уровня назад и продолжить игру. Отрицательный ответ (N) позволяет начать игру сначала.

Выключить силовое поле двери можно переключением четырех тумблеров. Их надо отыскать на данном уровне и определить правильную последовательность их включения. При правильном их положении под индикатором SWITCHES должны загореться четыре зеленых лампочки - поле отключено и можно идти. Тем не менее будьте осторожны - некоторые поля отключаются лишь на несколько секунд, что достаточно для входа, но для выхода придется переключать тумблеры еще раз.

Силовые поля, боевые роботы, мины - это еще не все неприятности, подстерегающие Вас. Наиболее серьезным препятствием является боевая механизированная установка, стреляющая самонаводящимися ракетами. Поразить ее без бомбы невозможно, но можно выстрелом из бластера изменить направление полета ракеты. Ракеты, не найдя нужной цели, самоликвидируются и Вы можете постараться как-то использовать это в своих целях.

На многих уровнях Вы встретите в красных квадратиках стрелки, показывающие определенное направление. Это транспортеры. Если Вы пойдете на стрелку, то пройдете сквозь нее, но назад здесь же вернуться не сможете, так что будьте внимательны.

Полезные советы.

1. Не играйте с партнером в перестрелку. Во-первых, это непроизводительные потери личной энергии, а во-вторых, при столкновении Ваших снарядов происходит блокировка этого места и Вы не сможете его пройти.

2. Как пройти вдвоем через дверь, если пропуск есть только у одного героя?

Делайте так: Герой, у которого есть пропуск, проходит через эту дверь и поджидает пока партнер не окажется напротив двери. Затем он подходит к двери, она открывается и пропускает партнера.

3. Главное правило: работу в отсеках космонавты выполняют по одиночке, а имущество собирают поровну. Если в комнате находятся два пропуска, то Вам надо взять один, а второй на всякий случай оставить партнеру. Если же пропуск всего один, значит кто-то из Вас должен его использовать, в то время, как второй космонавт должен находиться снаружи и страховать напарника, чтобы быть готовым в любой момент прийти к нему на помощь.

FORUM

Нам кажется, что программа ELITE столь же неисчерпаема, как и сам "ZX-Spectrum". Вот и сегодня пришло письмо, которое, возможно, откроет новые перспективы для тех, кто уже несколько раз прошел программу от начала до конца и ищет, чем бы теперь заняться.

Наш читатель из Киева Руслан Хоминич провел исследование трех разных версии игры, нашел в них существенные отличия и, если Вы хорошо освоили одну из версий, Вам может быть интересно попробовать свои силы и в другой.

Версия 1.

Программа вскрыта командой "JOYSTICK CLUB". После загрузки появляется надпись "Press SPACE Commander" и изображается корабль "ASP MK II".

Это самая легкая версия игры. Ваш корабль довольно активно атакуется, независимо от Вашего правового статуса, но погибнуть довольно сложно.

В этой версии сравнительно неэффективны ракеты - они уничтожают не более половины заряда энергетического отсека и, наверное, поэтому многие пилоты их не любят, считая дорогим и малоэффективным оружием.

В этой версии можно даже брать на таран корабли противника, причем статус FUGITIVE Вы за это не получите, даже если протараните мирный корабль.

Не встречается корабль ADDER.

Очень велики расстояния между планетами, топлива едва хватает для перелетов внутри круга, а на периферии круга расстояние уже слишком велико для перелета в один ход. Это же мешает использовать в полной мере клавишу F для перехвата таргоидов в гиперпространство, т.к. после возврата обычно не хватает топлива на обратный путь.

Атака на станцию не дает видимых результатов - станция просто не принимает Вас без каких либо объяснения. Вблизи станции корабли не встречаются.

Эту версию стоит рекомендовать начинающим пилотам.

Меняйло Е.В. из Калуги уточняет, что в этой версии самое короткое расстояние между соседними планетами составляет 3,2 св. года, подтверждает отсутствие корабля ADDER и добавляет, что никак не проходит заправка топливом от звезды.

Версия 2.

Программа вскрыта командой "TENARK SOFTWARE".

Главное отличие - отсутствуют корабли KRAIT. На заставочной картинке представлен корабль ADDER. В связи с этими обстоятельствами, в программе изменилась тактика пиратов.

В целом программа представляет средний уровень сложности.

Как и в предыдущей версии здесь ракеты малоэффективны. При попытке атаковать станцию, Вам выдают угрожающую надпись и Вы лишаетесь возможности стыковки. Станция оснащена системой ECM.

Эта версия позволяет стать миллионером в галактике 47 с помощью SAVE.

В обеих версиях имеют место захваты корабля пиратами при стыковке со станцией. Есть гипотеза, что Ваш корабль захватывается пиратами как бы "изнутри" в результате того, что Вы подбирали в космосе контейнеры с "рабами" и перегрузили свой корабль. Во всяком случае, не отмечалось захватов корабля, если в трюме не было "рабов".

Эта гипотеза имеет и то косвенное подтверждение, что были неоднократно отмечены захваты корабля в космическом пространстве "изнутри" после выхода из гиперперехода (об этом пишет, например, К. Дзреев из Ростова на Дону).

Версия 3

Версия имеет сообщение "M128".

Это наиболее сильная версия, она также наиболее соответствует фирменной инструкции.

Например, при попытке атаковать станцию, Вам навстречу вылетает стая полицейских кораблей и, хотя они не очень хорошо вооружены, их слишком много, чтобы устоять. Не спасает и перелет на другую планету, отныне в этой галактике Вы - изгнанник.

Значительно усложнена стыковка. Вокруг станции постоянно находятся различные корабли (преимущественно типа PYTHON) и, если Вы замешкаетесь, то рискуете с кем-нибудь столкнуться перед самой стыковкой. Часто это приводит к гибели.

Не менее опасно и резко тормозить после вылета со станции.

В этой версии большой разброс цен на некоторые запрещенные к перевозке товары, то есть Вас стимулируют к рискованным операциям. За Вами внимательно наблюдают. Если в глубинах космоса Вы совершите преступление, Вас могут не пустить потом на станцию. Здесь очень неприятно получать статус FUGITIVE. Не открывайте огонь первым, пока не увидите модель корабля визуально или пока он сам не начнет стрельбу.

Корабли типа KRAIT, THARGOID и SIDEWINDER здесь всегда пираты (возможно и ADDER, но пока он не встречался).

С кораблями типа PYTHON надо вести себя аккуратно. После первых выстрелов он выпускает KRAIT и SIDEWINDER - с ними можете воевать, это Ваша добыча, но головной корабль лучше оставить в покое и побыстрее покинуть поле боя.

FER-DE-LANCE - желательная добыча. Денег за него не дают, но рейтинг хорошо нарастает.

Нередко попадаются отшельники на астероидах. После обстрела они либо уходят, слабо маневрируя, либо отстреливаются, либо выпускают корабли KRAIT или SIDEWINDER. Из них можно "выжать" и контейнер с грузом, но поскольку отшельники защищены всегалактическим правом, Ваш статус может резко измениться.

В этой версии программы весьма эффективны ракеты. Попадание почти полностью выбивает защиту. Первым делом надо покупать систему ECM. Точно так же опасен и таран.

Сложность игры нарастает по мере повышения Вашего рейтинга. Сражения в галактике 1 намного проще, чем во второй галактике. Если в первой галактике Таргоиды поражались даже пульсирующим лазером, то во второй справиться с ними удалось далеко не сразу. Приходится иногда перестраивать тактику боя.

Фокус с 47-ой галактикой здесь не проходит.

Интересно, что отгрузочный блок имеет длину 104 байта. Видимо, это сделано для того, чтобы нельзя было использовать отгрузки с более легких версии.

Высокие качества этой версии подчеркивает также Сергей Дегтярев из Луганска. К сказанному выше мы можем добавить из его бортжурнала то, что при слишком высоком статусе FUGITIVE на станциях не открывается входное отверстие.

Кроме этих версии есть и другие, например версия, помеченная Jack O'Lantern. В ней стоит отметить отсутствие корабля KRAIT (сообщает Меняйло Е.В.). Условно назовем ее ВЕРСИЯ 4.

Лешинский А.М. (Н.Новгород) предполагает, что всего разных версии может быть порядка десятка, очень советует ввести какую-то систематизацию этих версий и упоминает о версии Родионова (назовем ее ВЕРСИЯ 5) и о версии Be-Be Soft (пароль 7Q) - назовем ее условно ВЕРСИЯ 6. К сожалению, он не приводит характерных особенностей данных версий.

"ИНФОРКОМ" горячо поддерживает предложение систематизировать версии и просит присылать свои предложения и идеи. В то же время, необходим какой-то универсальный параметр, который мог бы быть принят за основу классификации. Все читатели в письмах указывают на сообщения, сопровождающие или заканчивающие загрузку, но может быть целесообразно для каждой версии давать и раскладку по длинам входящих файлов?

*   *   *

У нас были вопросы от желающих получить дисковую версию программы ELITE - для них мы даем следующее сообщение:

Вышлю всем желающим дисковую версию ELITE ("Joystick Club") с записью состояния игры на диск.

169740, Коми, Усинский р-н, п. Приполярный, а/я 212, Сайфутдинов Е.В.

Сагдеев Р.Р. из г. Магнитогорска тоже работал с дисковой версией, вскрытой JOYSTICK CLUB, но в последнее время перешел на версию, в которой во время загрузки появляется надпись "M1 LOADING". Эта версия выглядит намного интереснее, отличается тем, что в ней есть очень сильные пираты и во много раз более серьезная полиция. Может быть, это что-то похожее на ВЕРСИЮ 3 по классификации Хоминича.

Перебросить ее на диск по блокам удается несложно с помощью копировщика PCOPIER, AMCOPY или каким-либо другим способом, загрузчик может быть таким:

```
10 BORDER NOT PI: PAPER NOT PI: CLS: CLEAR VAL "24751": RANDOMIZE USR VAL "15619": REM: LOAD
      "elite 1" CODE
20 RANDOMIZE USR VAL "15619": REM: LOAD "elite 2" CODE
30 RANDOMIZE USR VAL "24792": RANDOMIZE USR VAL "15619": REM: LOAD "elite 3" CODE VAL
      "16464"
40 REM POKE 46848,201
50 RANDOMIZE USR VAL "24795"
```

К сожалению, этого недостаточно, чтобы и отгрузка отложенного состояния игры и подгрузка тоже производились с дисковой поддержкой. Сагдееву Р.В. мешает это сделать отсутствие информации по управлению дисководом из машинного кода. Кстати, наш корреспондент отмечает, что адреса USR для старта блоков отличаются на 3. Это характерно для программ, сопровождающихся при загрузке надписью "M1 LOADING".

Перелеты к двойным звездам и невидимым звездам.

Мы уже писали о том, что в некоторых галактиках были обнаружены двойные звезды. Как уже было установлено, невидимые звезды - это тоже двойные звезды. К сожалению, не ко всем из них есть возможность слетать по желанию.

Меняйло Е.В. разработал схему перелета, основанную на безтопливном перелете со станции на станцию, о чем мы уже писали. (См. N11-12 "ZX-РЕВЮ-91", с. 253). В системе из пары звезд он называет одну основной, а вторую - подчиненной и предлагает следующий порядок действий.

1. Перелетите на основную планету любым способом.

2. Вызовите карту и с помощью поиска планеты по ее названию (клавиша "R") установите курсор на подчиненную планету. При помощи клавиши "P" это делать нельзя, т.к. информация имеется только об основной планете и при этом курсор будет всегда устанавливаться на основную планету.

3. Не нажимая клавишу "F" вылетите со станции и при помощи безтопливного перелета переместитесь на подчиненную планету. Оказавшись на ней, Вы можете просмотреть сводку цен, купить или продать товар, но при запросе информации Вам будут выданы данные только об основной планете.

Успешно посетил двойные звезды и наш читатель Владимир Кладов из Новосибирска. Он называет безтопливный перелет "перелетом-180", но пользоваться им не стал, а ввел в программу пару POKES.

POKE 60896,233

POKE 56272,233

Первый отключает поиск ближайшей планеты по команде "F", второй - по команде "H". Выдается информация (выполняется перелет) на планету, найденную предыдущей командой "O","R". Кстати, он приводит еще пару POKES.

POKE 56260,0 - "вечный межгалактический гипердвигатель".

POKE 28822,0 - "вечная энергетическая бомба".

Тайные возможности компьютера.

В эти апрельские дни, кажется решилась одна из важнейших проблем многочисленных любителей "Спектрума". Теперь каждый из Вас может, если захочет, в несколько минут конвертировать свой 48-ми килобайтный компьютер в полноценную 128-

килобайтную машину, причем сделает это абсолютно бесплатно.

Но, прежде чем мы перейдем к практике, несколько слов о сути.

Вы, конечно знаете, что Ваш Синклер-совместимый компьютер имеет 16 килобайтов памяти в ПЗУ и еще 48 килобайтов в ОЗУ, то есть всего 64K, а точнее 65535 байтов. Может он иметь больше? Наверное да, но ведь процессор Z-80 не всемогущ. Он работает с 16-разрядной адресной шиной и потому может обслуживать только 65535 байтов одновременно и не больше. Мы об этом писали в 1991 году на страницах "РЕВЮ", когда говорили о 128-килобайтных машинах. В них дополнительные 16-килобайтные блоки (называемые страницами) "впечатываются" на место страниц обычной памяти, подменяя их на время. Так что и в нем процессор работает всегда только с 64 килобайтами памяти.

А теперь рассмотрим такой вопрос. Вы, очевидно знаете, что Ваш компьютер в состоянии обслуживать еще и 65535 адресов внешних портов. Если Вы не работаете ни с какой периферией, то эта возможность просто никак не используется. А какая с точки зрения процессора разница - обслуживать внешние адреса или внутренние? Никакой. Вот если бы удалось так переключить процессор, чтобы он работал одновременно и с этими дополнительными адресами, как с оперативной памятью - Вы бы сразу имели 128 килобайтную машину без необходимости что-то паять и отлаживать.

Теперь мы Вас порадуем - такая возможность есть. Открыты два нигде не документированных регистра компьютера - раскрыта самая глубокая тайна К.Синклера, который уже в первых своих моделях предусмотрел их последующее расширение до 128K. Впрочем, нас интересует только первый регистр. Этот регистр называется первым Альтернативным Программным Регистром (сокращенно АПР.1). И за этим названием скрыта суть. Пожалуйста не путайте с альтернативным набором регистров процессора. Этот регистр к процессору не имеет никакого отношения и конфигурируется программным путем в результате серии сложнейших последовательностей команд IN и OUT.

Чтобы не забивать Вам голову, мы просто привели программу в машинных кодах, которая выполняет всю эту работу сама. К тому же мы еще сами не до конца разобрались, как же это все работает.

Наберите эту программу, запустите и наслаждайтесь грандиозным эффектом. До тех пор, пока компьютер не будет выключен, Вы можете чувствовать себя ничуть не хуже, чем любой владелец дорогой128-килобайтной машины, во всяком случае с этого момента жить Вам будет веселее.

Правда, после запуска этой программной последовательности, Вы не сможете получить того исходного меню, которое есть в стандартных 128-килобайтных "Спектрумах", но с этим ничего не поделаешь - ведь ПЗУ у Вас осталось от 48-килобайтной машины. Хотя, если подойти к задаче творчески, мы уверены, что кому-то из Вас удастся развить идею и получить меню.

Первая часть программы представляет обычный БЕЙСИК-загрузчик, который загружает машинный код, начиная с адреса 32768 и запускает его с адреса 32786. При загрузке проверяется также правильность Вашего набора кодов, которые хранятся в строках DATA.

После запуска Вам придется немного подождать (секунду-другую), пока программа "прощупает" архитектуру Вашей машины и выполнит необходимые операции.

Желаем успеха!

```
10 CLEAR 32767
20 LET x=32768
30 FOR i=0 TO 14
40 LET c=0
50 FOR j=1 TO 6
60 READ a
70 POKE x,a
80 LET x=x+1
90 LET c=c+a
100 NEXT j
110 READ a
120 IF a<>c THEN PRINT "ERROR IN LINE "; 10*i+160: STOP
```

```
130 NEXT i
140 RANDOMIZE USR 32786
150 STOP
160 DATA 22,0,0,17,7,16,62
170 DATA 3,65,80,82,73,76,379
180 DATA 32,49,45,83,84,32,325
190 DATA 62,2,205,1,22,1,293
200 DATA 7,0,17,0,128,205,357
210 DATA 60,32,6,64,197,1,360
220 DATA 11,0,17,7,128,205,368
230 DATA 60,32,193,16,243,17,563[1]
240 DATA 3,7,205,84,31,210,540
250 DATA 123,27,118,118,20,203,609
260 DATA 154,28,203,155,213,122,875
270 DATA 205,155,34,230,248,179,1051
280 DATA 33,0,88,17,1,88,227
290 DATA 1,255,2,119,237,176,790
300 DATA 209,24,219,0,0,0,452
```

Советы и секреты.

В ответ на просьбу о пароле к 8-му уровню игры IMPACT пришло немало писем от читателей. Так, Сельдемишев М.М. из г. Томска называет этот пароль - J.D. но ставит новый вопрос: - время от времени в этой программе в левом верхнем углу появляется буква "B" - что бы это значило? Рожков П.В. из Пензы не только дает этот пароль, но еще сообщает POKE 54500,183. Этот POKE уже есть в загрузчике программы, но стоит после оператора REM и потому не действует. Достаточно стереть REM и запустить программу. С другой стороны, у него есть просьба к обладателям игры TANTALUS (на кассетах ходит под названием TANTAL.KEY) - не может ли кто-нибудь помочь с POKES - игра хорошая, но пройти ее не удается. С помощью нашего трехтомника по программированию в машинных кодах ему удалось самостоятельно отыскать POKES в ряде программ и он рад ими поделиться:

FROST BYTE	33052,0 - жизнь
	33805,52 - время
DAN DARE 2	45891,0 - жизнь
SCEPTRE OF BAGDAD	58116,0 - жизнь
REBEL STAR 2	28599,N - кол-во ходов
PHANTOM CLUB	56486,0 - жизнь
TRUENO 1	25100,N - энергия
TRUENO 2	24785,0 - жизнь
	24984,N - энергия

Серию подсказок, достаточную для того, чтобы пройти всю игру SCEPTRE OF BAGDAD прислал наш читатель из Каракашев А.Г. из г. Грозный. Он очень интересуется адвентюрными играми, но, как и многие, не имеет возможности пополнять коллекцию в связи с тем, что они пока мало распространены.

1. Имея каску, можно пройти в комнату, где должна быть женщина, закрывающая ход в спальне. Теперь каску можно выбросить, больше никогда никто здесь не появится.

2. Флейта нужна, чтобы моток веревки при Вашем появлении в комнате поднимался вверх - тогда по ней можно подняться.

3. Имея крылья, идите к фонтану, ангел улетит на небеса, откроется подземный ход.

4. Огненного круг откроет люк (в фонтане).

5. Со связкой ключей идите в комнаты, расположенные за проходом, который раньше закрывала малоприятная личность, боящаяся каски. Дойдя до конца, прыгните в шкаф, ключи откроют потайной ход.

6. В фонтане трезубец можно обменять на жемчужину.

7. Имея сачок, идите в комнату с пчелой (пчелу предварительно надо выпустить из

---

[1] Скорее всего 561 (Прим. NUK)

гнезда). Поймайте пчелу и отнести ее в комнату с пауком. Паук исчезнет.

8. Шпагой можно отрубить веревку, на которой висел паук.

9. Жемчужиной, взятой в фонтане, зарядите рогатку.

10. Заряженной рогаткой сбейте кокосовый орех с пальмы, где сидит обезьяна.

11. Имея кокос можно идти в жаркую пустыню.

12. С книгой Али-Бабы можно открыть потайной код в подвале.

13. После этого можно в подвале наполнять кошелек золотом.

14. Имея кошелек, можно зайти в магазин и купить топор и башмаки.

15. Хлыстом усмирите быка и обменяйте его на ось.

16. Ось замените в подвале на сломанную.

17. Топором заточите кусок дерева.

18. С помощью зеркала убейте медузу.

19. Заточенным деревом убейте людоеда.

20. Веником усмирите волшебника и возьмите лампу, с которой можно смело идти за скипетром.

21. Одев башмаки, можно пройти через раскаленные угли.

22. Имея рыбу-шлем можно дышать под водой.

23. Веревкой натягивается лук.

24. Заряжается он стрелами.

25. С заряженный луком можно идти в комнату, расположенную за людоедом.

Далее уже совсем легко. Взяв жезл, возвращаетесь на балкон справа от исходной комнаты. Теперь мы видим счастливое лицо нашего героя, сжимающего в руках заветный скипетр.

Четыре предмета: песочные часы, тряпку, доспехи и перо применить нигде не удалось, но видимо они и не нужны, т.к. было набрано 100% очков.

Некоторые полезные советы:

1. Последовательность выполнения действия любая, удобная для игрока, правда, пройдя игру несколько раз, можно попробовать искать все более короткий путь. Конечно, при первом проходе трудно избежать бестолкового шатания по всему лабиринту туда-обратно.

2. Старайтесь никогда не делать себе безвыходных ситуаций, например, можно, забравшись по веревке вверх, умудриться забыть там флейту и спрыгнуть вниз без нее, или, например оставить в магазине мешок с золотом и выйти. Это может свести на нет все Ваши усилия.

3. Найдя бутылку, дающую OLD GAME не спешите тут же хватать ее, если у Вас еще достаточно жизней, а старайтесь сделать как можно больше действий с предметами, и взять бутылку, только когда жизней остается опасно мало.

ВНИМАНИЕ!

Дорогие друзья, сегодня мы можем помочь Вам стать непосредственными участниками новой, готовящейся в настоящее время телевизионной игры. Мы уполномочены на то, чтобы раскрыть перед Вами некоторые детали этого мероприятия и пригласить Вас к участию в небольшом конкурсе.

Ее устроитель, фирма "ФОНД", поставила перед собой интереснейшую задачу объяснить в форме популярной телевизионной игры что такое акции, что такое фондовая биржа, что такое дивиденд и как образуется курсовая разница. Зрители смогут наглядно увидеть и почувствовать, что такое фондовый рынок, узнают многое из того, что поможет им в будущем избежать возможных неприятностей, связанных с вложением денег в ценные бумаги.

"ИНФОРКОМ" участвует в подготовке этой телепрограммы на некоммерческой основе, обеспечивает ее компьютерную поддержку и, до некоторой степени, участвует в выработке ее концепции.

Начало регулярных трансляций телеигры в эфире запланировано на август-сентябрь месяц сего года. Названия телеигры мы Вам пока сообщить не можем, оно проходит регистрацию.

Теперь о сути.

Организация игры достаточно традиционна: пять игроков на сцене и несколько рядов зрителей в студии. Игроки получают начальный капитал (5000 рублей) и могут им распоряжаться, вкладывая деньги в акции тех или иных компаний. Акции можно не только покупать, но продавать если Вы чувствуете, что над компанией нависла угроза.

Всего в игре могут быть задействованы акции следующих двадцати компаний:

- судостроительная;
- лесоперерабатывающая;
- торговый дом;
- биржа;
- пищевая;
- авиационная;
- фармацевтическая;
- транспортная;
- топливно-энергетическая;
- газовая;
- пивоваренная;
- оборонная;
- банк;
- станкостроительная;
- золотодобывающая;
- прохладительных напитков;
- строительная;
- полиграфическая;
- автомобильная;
- химическая.

Финансовое положение этих компаний нестабильно и стоимость их акций непрерывно изменяется. Предугадать, как поведут себя акции той или иной компании можно только если очень внимательно читать газеты и делать при этом правильные выводы из прочитанных сообщений.

Перед каждым ходом игрокам дается какое-то газетное сообщение, которое может быть и совершенно шуточным и абсолютно серьезным, а уж они сами должны сообразить, как оно повлияет на стоимость акций тех или иных компаний.

Давайте рассмотрим, например вот такое сообщение:

"Как сообщают из высокопоставленных кругов, близких к правлению корпорации "Детский Мир", в течении двух-трех ближайших недель готовится десант Санта- Клаусов из штата Техас. В качестве ближней тактической задачи перед десантной группой ставится поздравление воспитанников московских детских домов и школ интернатов. Десант возглавляет президент компании "Шеврон". "

Такое сообщение может иметь, например следующие воздействия на рынок ценных бумаг:

Торговый Дом	+40%
Биржа	+20%
Фармацевтическая	+15%
Топливно-энергетическая	+22%
Газовая	+12%
Пивоваренная	-20%
Банк	+15%
Прохлад. Напитков	-25%
Автомобильная	-11%

Теперь рассмотрим, в чем здесь суть. В этом сообщении две ценных информации.

Во-первых, это привязка по дате. Очевидно, что речь идет о начале декабря месяца и приближается Рождество со всеми вытекающими отсюда последствиями.

Во-вторых (и это более сложно) те, кто внимательно следит за экономическими новостями, знают, что компания "Шеврон" подписала соглашение с Н. Назарбаевым о совместной эксплуатации нефтяных полей в Прикаспии. Эти нефтяные поля являются совместными для Казахстана и России и можно предположить, что гуманитарный визит президента этой компании в Москву может в будущем привести и к заключению аналогичных соглашений и с Россией.

Итак, акции Торгового Дома имеют всплеск, потому что в последний месяц перед Новым Годом совершается покупок больше, чем в последующие три месяца. Все хотят поздравить родных и близких. Кроме того, население, приученное к тому, что в новом году обязательно будут новые цены, спешит купить все, что запланировано, в последние недели.

Возрастание общего объема продаж влечет и повышение деловой активности на бирже и рост доходов биржи.

Временный подъем может иметь и фармацевтика, ведь в этот осенне-зимний период резко увеличивается количество вирусных заболеваний, а что делается в дни школьных каникул после многочисленных "Новогодних Елок" не надо объяснять.

Газовая промышленность имеет рост, поскольку приближаются зимние холода и требуется повышенный расход газа. То же относится и к топливно-энергетическому комплексу, но добавим еще ранее высказанные предположения о цели приезда президента компании "Шеврон".

Общий рост деловой активности несколько поднимает и акции банков.

Спад имеют автомобильная промышленность (спрос на автомобили в начале зимы традиционно меньше, чем в начале лета) и, конечно, сокращается потребление пива и прохладительных напитков.

* * *

Если хотите, можете попробовать свои силы в придумывании подобных сообщений. Присылайте то, что Вам удастся изобрести вместе со своими предположениями о реакции рынка на это сообщение.

Фирма "ФОНД" будет ежемесячно отбирать 7-8 лучших сообщений для включения их в последующие игры, а авторы этих сообщений получат приглашения в студию в качестве

зрителей. Иногородним фирма "ФОНД" обеспечит проживание и компенсацию затрат на проезд.

Кроме этого, среди приглашенных будет организован блиц-турнир. Его победитель получит право занять пятое игровое место на сцене и безвозмездно получит исходную сумму в 5000 рублей, необходимую для участия в игре. В ходе игры у него будет возможность ее увеличить или уменьшить.

После выхода в телеэфир презентационной передачи (август сентябрь) к участию в этом кон курсе подключатся миллионы телезрителей, но пока вы, наши уважаемые читатели, имеете неоспоримое преимущество.

Желаем Вам успеха!

Свои варианты присылайте пока на наш адрес, мы передадим их устроителям телепередачи - фирме "ФОНД". На конвертах делайте пометку "TV".

Уточняем наш адрес. 121019, Москва. Г-19, а/я 16.

Вниманию наших коммерческих представителей на местах, а также всем желающим подключиться к распространению программных продуктов "ИНФОРКОМа" для IBM-совместимых компьютеров мы представляем обучающий программный комплекс "БАРЬЕР".

I. НАЗНАЧЕНИЕ КОМПЛЕКСА

Комплекс предназначен для развития навыков скорочтения и, тем самым обеспечивает повышение интенсивности усвоения любых текстовых учебных материалов. Комплекс состоит из трех программ и имеет широкий спектр модификаций. Разработанный в период октябрь 1990 - март 1991г. , он реализуется нами на коммерческой основе уже более года. Количество продаж на сегодняшний день приближается к одной тысяче. В порядке подведения первых итогов скажем, что комплекс собирает самые благоприятные отзывы. Наиболее мощный эффект он имеет для детей любого возраста (умеющих читать).

Программы сами настраиваются на уровень подготовки пользователя, который с ними работает, не требуют никаких навыков по работе с ЭВМ, имеют ярко выраженный соревновательный характер и могут рассматриваться не только как тренирующие, но и как тестирующие и как игровые.

БАРЬЕР-1. Программа предназначена для развития навыков "фотографического" восприятия текстов и интенсивно тренирует фотографическую память.

БАРЬЕР-2. Программа предназначена для развития периферического зрения. С помощью тестовых таблиц производится диагностирование распределение концентрации внимания по зрительному полю.

БАРЬЕР-3. Тренирует блочное восприятие текстов при неполной информации (когда левая и правая часть поля страницы не охватываются периферическим зрением).

II. КОМПЛЕКТ ПОСТАВКИ

Комплекс поставляется на трех дискетах 5,25" (MS DOS, 360 K). Каждая программа на отдельной дискете.

III. МОДИФИКАЦИИ КОМПЛЕКСА

Программы комплекса "БАРЬЕР" дают проверенный эффект независимо от того, кто с ними работает, но тем не менее, мы подготовили ряд параллельных модификаций, рассчитанных на то, чтобы учебно-тренировочные тексты, используемые в программах, были по-возможности максимально приближены к конкретной сфере деятельности нашего заказчика.

Это следующие модификации:

БАРЬЕР-М - менеджмент и маркетинг

БАРЬЕР-В - вычислительная техника и программирование.

БАРЬЕР-Д - для детей (младший школьный возраст). Эта версия адаптирована для детей в смысле особого подхода к подбору учебных текстов. Как показывает опыт эксплуатации, одновременно с значительным (в 2-3 раза за один месяц) возрастанием скорости чтения у детей развивается память и правописание. Эта версия пользуется наиболее широкой популярностью.

БАРЬЕР-П - автоматизация производства и проектирования в машиностроении.

БАРЬЕР-Т - металлообработка (технология машиностроения).

БАРЬЕР-К - металлообработка (станки и инструмент).

БАРЬЕР-Р - радиотехника и электроника

БАРЬЕР-С - сельское хозяйство (растениеводство).

БАРЬЕР-О - общекультурное содержание. Эту версию заказывают при неопределенной профессиональной ориентации, например для школьников старших классов или для тех, кто связан с гуманитарными областями деятельности. Может быть

рекомендована клубам, кружкам и т. п.

ПРИМЕЧАНИЕ: Если в заказе не указано конкретно, какая версия необходима заказчику, мы поставляем версию БАРЬЕР-В (вычислительная техника и программирование).

IV. СРОК ИСПОЛНЕНИЯ ЗАКАЗА.

Срок исполнения - 2 - 3 недели после поступления средств на наш р/с.

V. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К АППАРАТНО-ПРОГРАММНОМУ ОКРУЖЕНИЮ

1. Полная аппаратно-программная совместимость с IBM PC XT/AT. Надежность функционирования на отечественных модификациях не гарантируется и не обсуждается.
2. Наличие "жесткого" диска ("Винчестера") стандартного объема.
3. Дисковод гибких дисков 5,25".
4. Операционная система - MS DOS не ниже 3.20.
5. Русификация компьютера в стандарте гост (кодировка альтернативная).
6. Требования к монитору - не специфицируются. Желательно - EGA.

VI. ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА

Гарантийным свидетельством при поставке программного продукта является картонный альбом, в который вложены дискеты с указанной на нем датой продажи При его отсутствии поставка выполняется с заверенным гарантийным талоном.

Гарантиями обеспечивается:

- бесплатная замена поставочных дисков, неработоспособных в состоянии поставки (в течение меся после поставки);

- замена с минимальной оплатой при выходе программы из строя по вине пользователя (механическое или электромагнитное повреждение, поражение вирусом на машине пользователя и т. п.) или по истечении месяца после поставки. Минимальная оплата не превышает стоимость дисков + 5% текущей стоимости программного обеспечения + стоимость почтово-транспортных расходов и согласовывается с потребителем.

VIII. ПОРЯДОК ОФОРМЛЕНИЯ ЗАКАЗА.

а) направить в наш адрес письмо-заказ с указанием необходимого программного продукта и количества копий. Приложить копию платежного поручения.

Крайне желательно сообщать также номер банковского АВИЗО. Это позволяет проводить розыск перечисленных средств в случае их длительной задержи в структурных подразделениях центрального Банка.

Наш адрес: 121019, Москва, Г-19, а/я 16, "ИНФОРКОМ"

б) произвести предварительную оплату платежным поручением на наш р/с:

N 500461778 во Фрунзенском коммерческом банке г. Москвы. МФО 201412.

Стоимость комплекса на период май - август 1992г.

БАРЬЕР-М - 1100 руб. + 28%
БАРЬЕР-В - 1100 руб. + 28%
БАРЬЕР-Д - 1100 руб. + 28%
БАРЬЕР-П - 1100 руб. + 28%
БАРЬЕР-О - 1100 руб. + 28%
БАРЬЕР-Т - 1300 руб. + 28%
БАРЬЕР-И - 1300 руб. + 28%
БАРЬЕР-Р - 1300 руб. + 28%
БАРЬЕР-С - 1300 руб. + 28%

"ЗЕЛЕНЫЙ ПАКЕТ" ДИСТРИБУТОРА

О том, что такое "зеленый пакет" Вы можете подробно прочесть в N11-12 "ZX-РЕВЮ" за 1991 г.

Стоимость "зеленого пакета" по комплексу "БАРЬЕР" составляет для частных лиц 430 рублей. Высылается наложенным платежом. Никакой предоплаты делать не надо, достаточно прислать заявку.

Напоминаем, что затраты дистрибутора на *зеленый пакет" являются только залогом и возвращаются по требованию. Активно работающим дистрибуторам затраты возвращаются при выплате комиссионных и далее такие пакеты предоставляются бесплатно.



МКП "ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

Уважаемые читатели!

Мы хотели бы обратиться к Вам со следующей информацией. Просим ею руководствоваться в дальнейшей деятельности.

1. Мы определились в наших планах на 1993 год и продолжим выпуск "ZX-РЕВЮ" в том же виде, т.е. это останется все такое же внутреннее "фирменное" издание для своих клиентов. Правда, по понятным экономическим причинам тираж его будет еще более сокращен.

2. Мы не сможем принимать от Вас подписку по почте так, как это делалось в прошлые годы и организуем в Москве корреспондентский пункт. Подписаться смогут только те, кто лично придут по известному им адресу или пришлют своих друзей, находящихся в Москве проездом. Подробности о том, как связаться с корпунктом - в прилагаемом к данному выпуску информационном листке.

3. В порядке исключения для жителей особо удаленных районов, не имеющим никакой возможности прибыть на корпункт мы рассмотрим проведение подписки по почте, хотя гарантируем это не всем. Будет отдано безусловное предпочтение нашим постоянным клиентам и авторам.

4. На корпункте кроме подписки на 1993 год можно будет приобрести выпуски 1992 года, полный комплект 1991 года, выполненный в виде отдельной книги улучшенного качества, и прочие сопутствующие материала.

5. Стоимость подписки и прочих материалов, при приобретении их через корпункт будет ниже, чем при аналогичном обслуживании по почте.

6. Мелкооптовые операции будут обеспечены дополнительными льготами.

7. Мы заключили соглашение о сотрудничестве с редакцией журнала "МОНИТОР", согласно которому часть материалов этого многотиражного журнала, посвященная особо сложным и интересным игровым программам для IBM-совместимых машин будет готовиться нами. Начиная с сентября месяца этого года (N5) и далее ежемесячно наши материалы будут печататься в этом журнале. Рекомендуем это издание Вашему вниманию как для подписки, так и для размещения рекламы.

Журнал "Монитор":
Тираж 30 000 экз.
Периодичность в 1992 году - 9 номеров в год.
Издатель: НТО "Софт-Москва".
Основная форма распространения розничная продажа, но подписка по почте возможна.
Адрес для писем: 117419, Москва, а/я 765. Тел. 247-36-25; 237-21-36.

* * *

Сообщаем вам график выхода очередник выпусков ZX-РЕВЮ этого года (по началу рассылки): N7-8 - 20 сентября N9-10 - 30 октября N11-12 - 15 декабря

До свидания, до встречи в следующем выпуске.

Ваш ИНФОРКОМ.

СПЕКТРУМ В ШКОЛЕ

В прошлом выпуске "ZX-РЕВЮ" мы напечатали тестирующую программу, которая может быть полезной на уроке истории. Сегодня мы приводим рекомендации по тому, как можно использовать компьютер на уроке географии.

Перед учащимся появляется на несколько секунд карта какой-либо страны (области, края) с указанным расположением городов. Затем города исчезают и остается только контур страны. Учащийся должен курсором указать где находится тот или иной город.

Вы можете сами задать ту или иную карту (в нашем примере рассмотрена Австралия). Это может быть, например Красноярский край.

Не надо думать, что эта программа применима только для определения месторасположения городов. С тем же самый успехом можно проверять знание учащимся основных районов добычи тех или иных полезных ископаемых, знание расположения горных хребтов, названий рек, расположения гидрокомплексов и т.п. Вы сами легко разберетесь, как Вам развить и применить эту программу.

Преподаватели биологии смогут применить аналогичный подход для проверки правильности знания учащимся названий различных частей растений или скелета животного. Возможностей много. Мы надеемся, что Вы сумеете ими воспользоваться.

```
10 REM Урок географии
20 LET another = 150:
      LET nextcity = 300:
      LET tryagain = 380:
      LET move = 400:
      LET wait = 420:
      LET finish = 580:
      LET outline = 1000:
      LET printcity = 2500:
      LET test = 2700:
      LET data = 3000
30 REM Инструкции обучаемому
40 BORDER 1: PAPER 7: INK 9: BRIGHT 1: CLS
50 PRINT PAPER 1; FLASH 1; AT 9,9; "УРОК ГЕОГРАФИИ"
60 PAUSE 150 70 CLS
80 PRINT AT 2,3; "Эта программа проверит Ваши знания по географии Австралии. На десять
      секунд Вам будет показано расположение австралийских городов. Потом они исчезнут и
      останется только контур. Ваша задача установить указатель в том месте, где должен
      находиться заданный Вам город. Вы имеете по три попытки на отыскание каждого города."
90 PRINT AT 19,5; FLASH 1; "Нажмите любую клавишу"
100 PAUSE 0 110 CLS
120 PRINT AT 2,2;
      "Клавиша 8 - указатель вправо
      Клавиша 5 - указатель влево
      Клавиша 7 - указатель вверх
      Клавиша 6 - указатель вниз
      Нажмите 0, когда указатель
      займет правильное положение
      После каждой попытки курсор
      будет возвращен в левый
      нижний угол."
150 PRINT FLASH 1;AT 19,5; "Нажмите любую клавишу"
140 PAUSE 0
150 REM another
160 DIM r(4)
170 LET error = 0
180 GO SUB outline
190 RESTORE data
200 FOR n=0 TO 6
210 READ a, b, Y, X, a$
220 GO SUB printcity
```

```
230 NEXT n
240 RESTORE data
280 PAUSE 500
290 GO SUB outline
300 REM nextcity
310 LET xcity = 0: LET ycity  = 0: LET tries = 1
320 LET s$ = "          ": REM десять пробелов
330 PRINT AT 1, 1; s$
340 PRINT AT 18, 1; "Найдем-"; AT 19, 1; s$
350 READ a, b, Y, X, a$
360 IF a$ = "eof" THEN GO TO finish
370 PRINT AT 19, 1; a$
380 REM tryagain
390 PRINT AT 1, 1; "attempt "; tries
400 REM move
410 PLOT OVER 1; xcity, ycity
420 REM wait
430 LET fall = 0: LET result = 0
440 IF INKEY$ = "0" THEN GO SUB test
450 IF result = 2 THEN LET r(tries) = r(tries)+1: PRINT FLASH 1;AT 20,1; "CORRECT": PAUSE
     25: PAUSE 150: PRINT AT 20,1; s$: GO SUB printcity
460 IF tries = 4 THEN LET r(4)=r(4)+1: LET error = 1: GO SUB printcity
470 IF result = 1 OR tries = 4 THEN GO TO nextcity
480 IF fail = 1 OR tries = 4
490 LET dx=(INKEY$="8")-(INKEY$="5")
500 IF xcity+dx=256 OR xcity+dx=-1 THEN LET dx=0
510 LET dy=(INKEY$="7")-(INKEY$="6")
520 IF ycity+dy=176 OR yclty+dy=-1 THEN LET dy=0
530 IF dz=0 AND dy=0 THEN GO TO wait
540 PLOT OVER 1; xcity, ycity
550 LET xcity=xcity+dx
560 LET ycity=ycity+dy
570 GO TO move
580 REM finish
590 CLS
600 PRINT AT 4,2; "Верно с первой попытки:   ";r(1)
610 PRINT AT 7,2; "Верно со второй попытки:  ";r(2)
620 PRINT AT 10,2;"Верно с третьей попытки: ";r(3)
630 PRINT AT 13,2;"Неверно: ";r(4)
640 INPUT "Попробуем еще раз?",y$
650 IF CODE y$=89 OR CODE y$=121 THEN GO TO another
660 STOP
950 REM*****************
     *               *
     *  Подпрограммы  *
     *               *
     *****************
1000 REM контуры
1010 CLS
1020 PLOT 51,58
1030 DRAW 0,4
1040 DRAW 2,0
1050 DRAW 0,9
1060 DRAW -15,31
1070 DRAW 4,-3
1060 DRAW 1,2
1090 PLOT 51,128
1100 DRAW -8,-27,1.5
1110 PLOT 51,128
1120 DRAW 3,-1
1130 DRAW 6,3
1140 DRAW 12,9,1.5
1150 DRAW 3,1
1160 DRAW -1,4
1170 DRAW 3,5
```

```
1180 DRAW 3,-6
1190 DRAW 2,3
1200 DRAW -2,2
1210 DRAW 1,2
1220 DRAW 4,-2
1230 DRAW 0,7
1240 DRAW 10,7
1250 DRAW 3,-1
1260 DRAW 3,5
1270 DRAW 4,-2
1280 DRAW -2,4
1290 DRAW 7,9
1300 DRAW 8,1,1
1310 DRAW 0,2
1320 DRAW 2,1
1340 DRAW 1,-2
1350 DRAW 19,0,1
1360 DRAW 2,-2
1370 DRAW -7,-13
1380 DRAW 21,-14
1390 DRAW 4,1
1400 DRAW 5,11
1410 DRAW 2,20
1420 DRAW 3,1
1430 DRAW 1,-6
1440 DRAW 2,-4
1450 DRAW 1,-9
1460 DRAW 4,1
1470 DRAW 0,-2
1480 DRAW 3,-3
1490 DRAW 0,-7
1500 DRAW 2,-2
1510 DRAW 2,-10
1520 DRAW 2,-1
1530 DRAW 11,-13
1540 DRAW 0,-5
1550 DRAW 3,0
1560 DRAW 0,2
1570 DRAW 3,-1
1580 DRAW 0,-4
1590 PLOT 210,71
1600 DRAW -6,37,1.5
1610 PLOT 210,71
1620 DRAW -15,-26
1630 DRAW -3,-6
1640 DRAW -2,-1
1650 DRAW -8,-3,1.2
1660 DRAW -2,-1
1670 DRAW -1,-3
1660 DRAW -2,3
1690 DRAW -7,4
1700 DRAW -5,-3
1710 DRAW -11,4
1720 DRAW -2,5
1730 DRAW 1,1
1740 DRAW -5,7
1750 DRAW -2,-1
1760 DRAW 0,8,1
1770 DRAW -3,-5
1780 DRAW -1,5
1790 DRAW 3,3
1800 DRAW -2,3
1810 DRAW -9,-9
1820 DRAW -50,5,2.4
1830 DRAW -13,-2
```

```
1840 DRAW -6,-3
1850 DRAW -7,-1
1860 DRAW  6,4
1670 DRAW -1,-1
1880 PLOT 213,96
1890 DRAW 2,4,.5
1900 PLOT 142,55
1910 DRAW -2,0
1920 DRAW -1,-2
1930 DRAW 2,0
1940 DRAW 1,2
1950 PLOT 170,25
1960 DRAW 15,-1,7
1970 DRAW 1,2
1980 DRAW -1,3
1990 DRAW 3,-2
2000 PLOT 183,9
2010 DRAW 5,18,.5
2020 PLOT 183,9
2030 DRAW -3,0
2040 DRAW 0,-2
2050 DRAW -4,0
2060 DRAW 0,1
2070 DRAW -2,0
2080 DRAW -1,10
2090 DRAW -3,4
2100 DRAW 0,4
2110 RETURN
2500 REM города
2510 PLOT FLASH error;a,b
2520 PLOT FLASH error;a,b+1
2530 PLOT FLASH error;a+1,b
2540 PLOT FLASH error;a+1,b+1
2550 CIRCLE FLASH error;a,b+1,3
2560 PRINT FLASH error;AT Y,X;a$
2570 IF error=0 THEN RETURN
2580 PAUSE 50: PAUSE 200
2590 LET error = 0
2600 GO TO printcity
2700 REM test
2710 PLOT OVER 1; xcity, ycity
2720 IF ABS (xcity-a) < 4 AND ABS (ycity-b) < 4 THEN LET result=1
2730 IF result = 0 THEN LET tries=tries+1:LET fall=1
2740 LET xcity = 0: LET ycity = 0
2750 PAUSE 25
2760 RETURN
2900 REM  ************
          *          *
          *  Данные  *
          *          *
          ************
3000 REM Данные по городам
3010 DATA 172,39,17,22,"Мельбурн"
3020 DATA 199,56,15,25,"Сидней"
3030 DATA 53,71,13,7,"Перт"
3040 DATA 147,55,13,14,"Аделаида"
3050 DATA 110,165,1,14,"Дарвин"
3060 DATA 179,10,20,23,"Хобарт"
3070 DATA 212,81,10,23,"Брисбейн"
3080 DATA 0,0,0,0,"eof"
```

BETA BASIC

Продолжение.
Начало см. стр. 3,47

16. DO

или DO WHILE <условие>
или DO UNTIL <условие>

Клавиша D (Ключевое слово WHILE находится на клавише J, а ключевое слово UNTIL - на клавише K).

См. также LOOP, EXIT IF.

Конструкция DO - LOOP имеет ряд преимуществ по сравнению с обычным способом организации циклов FOR - NEXT стандартного БЕЙСИКа. Эти преимущества становятся еще более ощутимыми при использовании квалификаторов WHILE и UNTIL.

Без них DO и LOOP являются просто маркерами, отмечающими начало и конец цикла. После того, как программа встретит оператор LOOP, управление передается на оператор DO и т.д.

Пример:

```
10 DO
20   PRINT "HELLO"
30 LOOP
```

Эта программа будет печатать бесконечное число раз слово HELLO. Остановить ее можно будет только нажав BREAK.

Действие DO можно изменить с помощью квалификатора WHILE <условие>. Его действие таково:

Если условие стоящее после WHILE справедливо (имеет значение "истина"), то выполняются операторы стоящие после DO до тех пор, пока не встретится оператор LOOP, после чего управление вновь передается на DO и вновь проверяется справедливость условия и т.д. Если же условие "ложно", то вся часть программы, стоящая между DO и LOOP игнорируется и управление передается к оператору, стоящему за LOOP.

Таким образом, та часть программы, которая стоит между DO WHILE <условие> и LOOP выполняется раз за разом, пока <условие> справедливо.

DO UNTIL <условие> имеет прямо противоположное значение. Часть программы заключенная между DO и LOOP выполняется, пока условие "ложно" (иными словами до тех пор, пока условие не станет справедливым).

Пример:

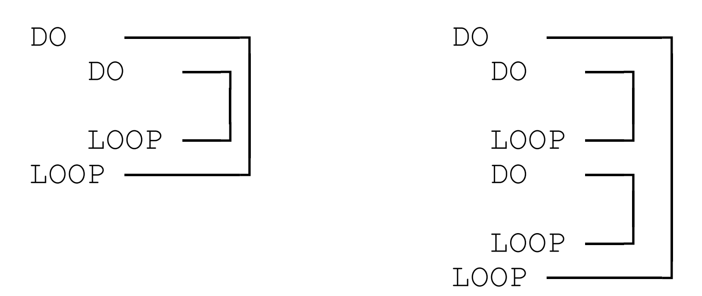
```
10 LET total = 0
20 DO UNTIL total > 100
30   INPUT "Введите число ";x
40   LET total = total * x
50   PRINT total
60 LOOP
70 PRINT "Получили число больше ста"
```

В этом примере строку 20 можно было бы заменить такой:

```
20 DO WHILE total < = 100.
```

Пары DO-LOOP могут вкладываться точно так же, как FOR - NEXT.

Например:



Компьютер запоминает адрес, по которому в программе хранится оператор DO. Для запоминания служит стек, поэтому нельзя выходить из цикла (перепрыгивать через LOOP), иначе как с помощью EXIT IF или POP. Иначе может произойти "закупоривание" стека и сбой работы в программе.

Если в программе для оператора DO не найден соответствующий LOOP, выдается сообщение об ошибке:

```
S, "Missing LOOP".
```

Аналогичные конструкции существуют и во многих других языках программирования и могут иметь другое написание, вот примеры аналогичных конструкций:

```
REPEAT
   операторы
UNTIL <условие>
```

аналогично:

```
DO
   операторы
LOOP UNTIL <условие>
REPEAT
   операторы
UNTIL FALSE
```

аналогично:

```
DO
   операторы
LOOP
WHILE <условие>
   операторы
ENDWHILE (ИЛИ WEND)
```

аналогично:

```
DO WHILE <условие>
   операторы
LOOP
```

17. DPOKE адрес, число

Клавиша: P

См. также DPEEK (адрес), число

DPOKE означает "двойной" POKE.

Этот оператор засылает двухбайтное число в две адресных ячейки. Аналогичная конструкция в стандартном Бейсике имеет следующий вид:

```
POKE a,n-INT(n/256)*256
POKE a+1,INT(n/256)
```

Здесь a - адрес

n - число (0...65535)

Другими словами, младший байт засылается по указанному адресу, а старший байт - в следующий за ним адрес. Поскольку многие системные переменные "Спектрума" имеют именно такой двухбайтный формат, то их очень удобно изменять с помощью DPOKE. Аналогично функция DPEEK представляет "двойной" PEEK.

18. DRAW TO x,y <,угол>

Здесь используются два ключевых слова стандартного Бейсика - DRAW и TO.

Этот оператор вычерчивает линию от текущей графической позиции до точки, заданной координатами x,y. Часто это более удобно, чем использовать стандартный DRAW, при котором задаются не абсолютные координаты, а величина относительного "смешения" по горизонтали и вертикали.

Пример:

```
10 FOR n=1 TO 100
20   DRAW TO RND*255,RND*175
30 NEXT n
```

Если Вы зададите и третий параметр (угол), то сможете изображать кривые.

После TO Вы можете поставить оператор цвета PAPER, INK или атрибут, например OVER и т.п.

```
DRAW TO 10,10
DRAW TO INK 2;20,30
DRAW TO 100,90,1
```

18. EDIT <номер строки>

Клавиша: 0 (в обычном, не графическом режиме).

Это не то же самое, что SHIFT + "1". EDIT - это ключевое слово. Оно предназначено для того, чтобы избежать утомительной последовательности: LIST - выбор строки - BREAK - SHIFT + "1".

Для того, чтобы быть по-настоящему полезным, EDIT должен вызываться без нажатия SHIFT-клавиши. Но все буквенные клавиши уже заняты, а цифровые клавиши нужны для ввода номеров строк. В то же время, номера строк не начинаются с нуля, поэтому эту клавишу можно использовать.

Вы закончили набор какой-либо строки. Нажали ENTER. Программа ждет ввода номера очередной строки. Вы нажимаете 0 и получаете ключевое слово EDIT (если даже оно и не появятся, все равно нуль в начале строки означает EDIT). Если после EDIT (или нуля) набрать номер нужной Вам строки и нажать ENTER, эта строка мгновенно будет помешена в нижнюю часть экрана для последующего редактирования.

Если номер строки не указан, то на редактирование поступит текущая строка. Дополнительно для упрощения редактирования введена возможность перемещать курсор по экрану "вверх"/"вниз" в рамках программной строки. Курсор не может вставать внутри ключевых слов, а занимает ближайший пробел. Если Вы попытаетесь поднять его выше, чем вершина строки или опустить ниже нижней строки экрана, он автоматически "прыгнет" в конец программной строки. Самый простой способ добавить что-то к концу строки - вызвать EDIT и затем нажать "курсор вверх".

9. EDIT строковая переменная

или EDIT ; числовая переменная

Клавиша: SHIFT + "5"

EDIT может применяться не только для удобного редактирования строк, но и для простого изменения программных переменных. Для этих целей нельзя использовать нуль в начале строки и введена возможность вызова EDIT нажатием SHIFT + "5" в графическом режиме. Можно также набрать слово EDIT по буквам в режимах KEYWORDS 3 или 4. Типичное применение - внесение изменений в строковые переменные, например для изменения фамилии, имени, отчества, адреса в массиве данных, представляющем из себя базу данных.

Такой массив мог быть заполнен, например с помощью INPUT. Если теперь надо в нем что-то изменить, то это трудоемкая задача для обычного БЕЙСИКа, а здесь с помощью EDIT a$(n) Вы получаете содержимое строки "n" в области редактирования. Рассмотрим пример:

```
10 LET a$ = "John Brown"
20 EDIT a$
30 PRINT a$
40 LET num=365.253
50 EDIT ; num
60 PRINT num
```

В строке 50 применен знак ";" для того, чтобы отличить режим EDIT <числовая переменная> от режима EDIT <номер строки>. Можно было бы, однако, вместо точки с запятой использовать и запятую, но в этом случае редактируемая переменная была бы изображена, начиная с 16-ой позиции экрана.

EDIT имеет синтаксис, очень похожий на INPUT. Вы можете использовать точку с запятой, запятую, AT, TAB, LINE и строковую подсказку точно так же, как обычно это делают с оператором INPUT. Отличие в том, что редактировать можно только одну переменную. Если Вы попробуете редактировать несколько, то все прочие, кроме первой, будут восприняты, как операторы INPUT. Нижеследующий пример показывает, как можно создавать массивы и редактировать их:

```
10 DIM a$(10, 15)
20 FOR n=1 то 10
30   INPUT a$(n)
40 NEXT n
50 PRINT "Редактирование"
60 FOR n=1 TO 10
70 EDIT ("запись ";n;" ");a$(n)
80 NEXT n
```

Если переменная, подлежащая редактированию в EDIT не существует, то команда полностью эквивалентна INPUT.

20. ELSE <оператор>

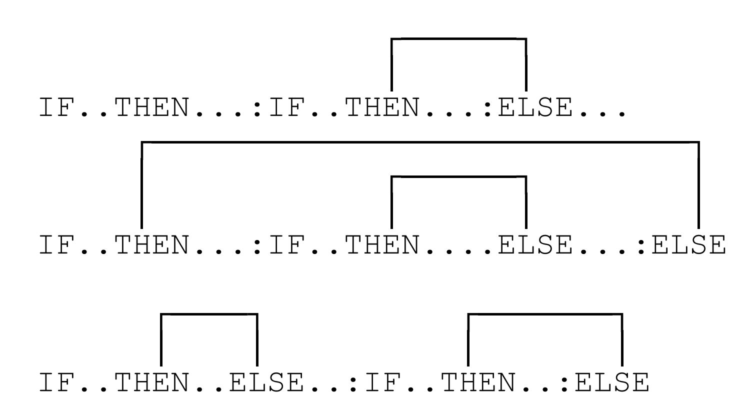
Клавиша: E

ELSE - один из элементов конструкции IF - THEN. Обычно, если условие, стоящее после if несправедливо (ложно), то следующей выполняется строка, стоящая за строкой, содержащей этот IF. Этот порядок можно изменить. Если IF и THEN содержат еще и ELSE, то в случае, когда условие не выполняется, управление передается оператору, стоящему после ELSE. С другой стороны, если условие справедливо, то строка IF исполняется только до ELSE и затем управление передается следующей строке (если THEN не передал его в иное место).

Например:

```
10 INPUT "Задайте число ";x
20 PRINT "Это число равно 1?"
30 PAUSE 50
40 IF x=1 THEN PRINT "ДА": ELSE PRINT "НЕТ"
50 GO TO 10
```

Однако, ситуация может быть осложнена, если на одной строке у Вас есть несколько IF THEN ELSE. Тогда может быть трудным определить, что же к чему относится. Приведенные ниже примеры помогут Вам разобраться.



Если ELSE используется без IF, например, когда ELSE стоит первым оператором строки, то и ELSE и остальная часть строки игнорируются.

21. END PROC

Клавиша: 3

См. также раздел, посвященный процедурам PROCEDURES; DEF PROC.

Этот оператор отмечает конец процедуры. Благодаря этому компьютер не будет выполнять ничего, стоящего между DEF PROC и END PROC, если процедура не была вызвана. Он просто перепрыгнет через нее за END PROC. Во время же исполнения процедуры, END PROC служит концом ее исполнения. Удаляются локальные переменные (LOCAL) восстанавливаются исходные значения глобальных (GLOBAL) переменных, если они есть. Если в процедуру какие-то параметры передавались по ссылке (перед формальным параметром стояло REF), то текущее значение формального параметра присваивается фактическому параметру, задействованному при вызове.

Если формальный параметр не найден, Вы получите сообщение "Variable not found".

После исполнения процедуры управление передается оператору, стоящему за вызовом процедуры.

Применение END PROC без соответствующего DEF PROC дает сообщение об ошибке:

`W, "Missing DEF PROC".`

22. EXIT IF <условие>

Клавиша: I

См. также DO, LOOP

Этот оператор является элементом конструкции цикла DO-LOOP (см. соответствующий раздел). EXIT IF используется для того, чтобы выйти по своему желанию из середины цикла DO-LOOP, если условие выполняется. После выхода из цикла управление передает оператору, следующему за LOOP.

Если условие не выполняется, ничего не происходит.

Пример:

```
100 DO
110   PRINT "строка 110"
120   PAUSE 20
130 EXIT IF INKEY$ = "" STOP
140   PRINT "строка 140"
150   PAUSE 20
160 LOOP
170 PRINT "Выход из цикла"
```

Программа будет работать до тех пор, пока не будет нажата клавиша "STOP" (SYM. SHIFT + A).

Если в программе опущен оператор LOOP, выдается сообщение ошибке:

S, "Missing LOOP"

23. FILL x,y

или FILL <INK число; > x,y

или FILL <PAPER число;> x,y

Клавиша: F

Команда FILL или FILL INK закрашивает область экрана, окрашенную цветом PAPER в цвет INK, начиная с координат x,y.

Команда FILL PAPER наоборот закрашивает область экрана, окрашенную цветок INK в цвет PAPER.

Закрашивание происходит от точки с координатами x,y во все стороны до тех пор, пока не встретятся участки, уже окрашенные в INK для команд FILL INK и FILL или в PAPER для команды FILL PAPER. Если исходная точка и ее окрестности окрашены в требуемый ничего не происходит.

В отличие от обычного Бейсика параметр цвета после INK или PAPER может не указываться, в этом случае идет закрашивание текущим цветом. Например, FILL PAPER; x,y - сработает.

Пример:

```
10 FOR n=1 TO 6
20    CLS
30    CIRCLE INK n;128,88,n*10
40    FILL INK n; 128,88
50 NEXT n
```

Возможно и использование сложных конструкций FILL, таких как:

```
FILL INK 2; PAPER 1; FLASH 1; X,Y
```

В таких случаях первое ключевое слово после FILL указывает на то что используется при заполнении (INK или PAPER), а остальные просто меняют атрибуты заполняемой области.

Поскольку, как Вы знаете, в пределах одного знакоместа невозможно использование более двух цветов одновременно, техника работы с цветом должна быть очень хорошо продуманной. Очень легко получить неприемлемый результат, особенно в тех местах, где контактируют два цвета INK. Обычно это обходят за счет того, что стык областей с разными цветами проводят по границам знакомест.

```
10 LET x=128, y=88, rad=70
20 LET g= (SQR 2)*rad/2
30 CIRCLE  x,y,rad
40 PLOT x,y : DRAW -g,-g
50 PLOT x,y : DRAW g,-g
60 PLOT x,y : DRAW 0,rad
70 FILL INK 2; x-5,y
80 FILL INK 4; x+5 y
```

FILL работает на областях любой геометрической формы. Для примера попробуйте заполнить весь экран, за исключением того, что находится внутри букв "Q".

```
PRINT STRING$ (704, "Q"): FILL 0,0
```

Метод работает быстро, если есть большое количество доступной свободной памяти. Если Вы заполняете большую область, например весь экран, то сможете увидеть очевидные паузы в те моменты, когда компьютер проверяет не забыл ли он окрасить какие-то участки. Можно рекомендовать делить сложную область на несколько более простых и заполнять их порознь.

Можно установить, какое количество пикселей было закрашено по последней команде FILL, для этого служит функция FILLED().

Прервать заполнение можно в любое время нажатием BREAK.

24. GET числовая переменная

или GET строковая переменная

Клавиша: G

Это то же самое, что GET от клавиатуры. Как и INKEY$ это способ чтения клавиатуры без использования ENTER. Отличие от INKEY$ в том, что GET ждет нажатия клавиши. При использовании со строковой переменной, GET вводит один символ:

```
10 GET a$: PRINT a$;: GO TO 10
```

Работа этой строки похожа на работу пишущей машинки. Обычным путем Вы можете переключаться на печать прописными и строчными буквами. Курсор можно перемещать в нужном направлении, работает стирание символов. ENTER дает переход к началу следующей строки. Более изощренная версия выглядит так:

```
10 GET a$
20 PRINT CHR$ 8; " "; CHR$ 8; a$; INVERSE 1; "B"; INVERSE 0;
30 GO TO 10
```

Может быть, Вы захотите изменить размер символов, воспользовавшись командой CSIZE.

Если GET применяется с числовой переменной, то при нажатии клавиш от "1" до "9" вводится число от 1 до 9. При нажатии "A" или "a" вводится число 10, при нажатии "B" или "b" вводится число 11 и т.д. Это очень удобно для организации разного рода экранных меню при написании меню-управляемых программ (см. также ON).

25. GET строковая переменная, x,y <ширина, длинах><;тип>

Это как бы GET с экрана. Применяется для того, чтобы какой-то прямоугольный блок экрана присвоить некоей строковой переменной и впоследствии печатать его на экране с помощью PRINT или PLOT в тех позициях экрана, в каких захотите.

(Если Вы будете применять PRINT, то необходимо учесть, что установка CSIZE не должна быть нулевой. Так, например, CSIZE 8 даст печать сохраненного в строковой

переменной изображения в натуральную величину.)

Размер задается в знакоместах, а координаты - в пикселах. В простейшем случае за именем строковой переменной следуют координаты левого верхнего угла.

```
10 PRINT "QWE"
20 GET a$,4,175
30 PLOT 100, 100; a$
```

Т.к. размеры снимаемого с экрана блока не указаны, то по умолчанию предполагается, что длина и ширина равны по одному знакоместу.

В этом примере на экране будет напечатано "QWE", а затем правая половина буквы Q и левая половина буквы W будут сохранены в переменной a$. С помощью усовершенствованной команды Бета-Бейсика PLOT Вы сможете напечатать полученное изображение в любом месте экрана (см. PLOT), причем сделать это можно с разным увеличением (см. CSIZE).

Здесь есть один нюанс. Как может строковая переменная содержать часть графики экрана? Если Вам это не интересно, пропустите несколько следующих абзацев.

Поставим маленький эксперимент. Узнаем длину строковой переменной a$. Это можно сделать PRINT LEN a$. Вы увидите, что длина этой переменной равна 9 символам. Первый содержит управляющий код, который говорит о том, что в последующих 8 символах идет графический образ. Если Вы сняли блок большего размера, а не одно знакоместо, то в него вставлены там, где это надо, коды управления курсором (см. раздел Управляющие Коды).

Нижеприведенный пример изображает на экране блок размером 3x3 знакоместа, запоминает его в переменной a$ и затем печатает в случайно выбранных позициях экрана, причем делает это гораздо быстрее, чем это можно было бы сделать повторным его перестроением.

```
10 CIRCLE 10,165,10
20 CIRCLE 13,165,5
30 FILL 5,165
40 GET a$,0,175,3,3
50 PLOT RND*230, RND*150+20; a$
60 GO TO 50
```

Если Вы используете OVER 0, то увидите, что поля рисунка затрут цветом PAPER то, что лежит под ними. Для получения других результатов попробуйте использовать OVER 1 или OVER 2 и запустите пример еще раз (OVER 2 это режим не стандартного БЕЙСИКа, а БЕТА-БЕЙСИКа, позволяющий Вам рисовать и без затирания и без инвертирования того, что лежит под Вашим изображением.)

Если Вы хотите нарисовать много окружностей одного размера, то гораздо быстрее работают GET и PLOT, чем CIRCLE. Попробуйте удалить строки 20 и 30 и введите OVER 2 (или используйте в строке 50 PLOT OVER 2; RND * ...)

Эти рассмотренные примеры относятся к нулевому типу. Если Вы не указываете при команде GET параметр <тип>, то предполагается, что он равен нулю. Нулевой тип команды GET - "бесцветный".

То есть рисунок снимается с экрана без цветовых атрибутов и воспроизводится командой PLOT в текущих установленных цветах или в локальных цветах, указанных в команде PLOT или PRINT.

Вы можете использовать команду GET и другого типа. Это тип-1. Для его использования после команды поставьте точку с запятой и поставьте 1. В этом случае изображение снимается с экрана в строковую переменную вместе с цветовыми атрибутами. Точно так же оно будет и изображено по команде PLOT или PRINT. Принципиально важно, что если графический блок, с которым Вы работаете, имеет несколько цветов, то все их можно будет воспроизвести только при работе с первым типом GET. Ни глобальные, ни локальные установки цвета не повлияют. Изображение будет воспроизведено так, как было снято.

```
10 PRINT INK 2; "ABC"
20 PRINT: PRINT INK 4; "DEF"
30 GET s$,0,175,3,3; 1
```

```
40 PLOT 100,100; s$
```

Как обычно, Вы не должны забывать, что для "Спектрума" в пределах одного знакоместа возможны только 2 цвета (INK и PAPER), поэтому тщательно планируйте, куда Вы помещаете изображение, снятое по GET вместе с цветами. При неаккуратной печати его на экране Вам не избежать проблем с атрибутами ("клэшинг" атрибутов).

26. JOIN <номер строки>

Клавиша: SHIFT + 6 (то же, что и "&")

См. также SPLIT.

Объединяются вместе две строки. Первая строка - номер которой задан (если не задан - то та строка, на которой стоит курсор) и следующая за ней строка.

Совместно используя SPLIT и JOIN Вы получаете возможность "отрезать" часть строки и "пристегнуть" ее к другой.

27. JOIN строковый массив или числовой массив.

Клавиша: SHIFT + 6 (то же, что и "&")

См. также главу "Обработка данных", с. 8.

Эта команда позволяет переместить часть символьной строки или массива в любую позицию другой символьной строки или числового массива. С командой JOIN тесно связана еще команда COPY. Поскольку они очень похожи друг на друга, то и обсуждаться здесь будут совместно. Разница их действия состоит в том, что если команда JOIN перемещает данные, то команда COPY - копирует их. Разница очевидна. При копировании исходный материал не уничтожается, а при перемещении он пропадает.

а) Работа с символьными массивами.

Образец синтаксиса:

```
JOIN a$ <n TO m> TO b$ <k>
COPY a$ <n TO m> TO b$ <k>
```

Здесь:

a$ - исходная символьная строка;

b$ - строка назначения;

n,m - параметры выделения подстроки из строки;

k - позиция в строке назначения.

Пример:

```
10 LET a$ = "12345"
20 LET b$ = "ABCDEFG"
30 JOIN a$ TO b$
40 PRINT b$: REM печатается ABCDEFG12345
50 PRINT a$: REM: a$ не найдено.
```

Поскольку мы применяли JOIN и не указывали параметров n,m, то вся переменная a$ целиком перешла в b$ и печать a$ в строке 50 ничего не даст. Если же теперь Вы в строке 30 замените JOIN на COPY, то сможете убедиться в строке 50, что a$ не пострадало. Вариант применения COPY, тем самым, похож на то же, что мы имеем просто работая с LET:

```
LET b$ = b$ + a$
```

Но зато и JOIN и COPY работают даже тогда, когда a$ и b$ такие огромные, что заполняют всю память компьютера, а LET в этом случае работать не сможет. Например LET a$ = a$ + "x" не сможет работать, если a$ длиннее, чем одна треть свободной памяти компьютера.

С помощью параметров выделения подстроки из строки Вы можете работать не только со строками, но и с подстроками. Их можно вставлять в строку назначения в заданное место, указав параметр позиции вставки.

Если не заданы параметры подстроки, то принимается по умолчанию вся строка. Если не задан параметр позиции вставки в строку назначения, то вставка выполняется после последней позиции. Попробуйте поэкспериментировать с вышеприведенным примером, заменяя строку 30:

```
30 JOIN a$ (2) TO b$
       REM:
       a$="1345",
       b$="ABCDEFG2"
30 JOIN a$ (3 TO) TO b$
       REM:
       a$="12",
       b$="ABCDEFG345"
30 COPY a$ TO b$(3)
       REM:
       a$="12345",
       b$="AB1234CDEFG"
30 JOIN a$(2 TO 3) TO b$(LEN b$+1)
       REM:
       a$="145",
       b$="ABCDEFG23"
```

А теперь рассмотрим конкретный практический пример. Вам надо просмотреть символьную строку t$ и всякий раз, как в ней встретится слово "Spectrum" заменить его на "ZX". Для этого можно исполь зовать COPY, DELETE и INSTRING.

```
10 LET t$="The Spectrum is a versatile computer, but Spectrum owners may feel it should have
       better editing"
20 PRINT t$
30 LET n$="ZX", o$="Spectrum"
40 LET P=1
50 LET P=INSTRING (P,t$,o$)
60 IF P<>0 THEN
       DELETE t$(P TO P+LEN o$-1)
       COPY n$ TO t$(P)
       GO TO 50
70 PRINT t$
```

b) Работа с символьными массивами.

Образец синтаксиса:

```
JOIN a <n TO m> TO b <k>
COPY a <n TO m> TO b <k>
```

Здесь:

a - исходный массив;

b - массив назначения;

n, m - параметры выделения подмассива из массива,

k - позиция в массиве назначения.

Массивы - это удобный метод работы с большим количеством данных, но обычно они имеют много серьезных ограничений. Наверное самое существенное из них - то, что массивы имеют фиксированный размер, объявленный в момент назначения массива. Вы можете нерационально расходовать память, назначив (DIM) массив больше, чем Вам на самом деле необходимо, и Вам может не хватать памяти например для создания еще одного массива.

БЕТА-БЕЙСИК позволяет более гибко манипулировать с массивами. Команды JOIN и COPY позволяют перемещать или копировать весь массив или его часть. Пространство для нового материала создается внутри назначенного массива, поэтому ничего не будет перезатерто. Можно обслуживать не только одномерные, но и двумерные массивы, а этого бывает достаточно для большинства приложений. Одномерные массивы трактуются, как двумерные, имеющие вторую размерность, равную единице. Нет никакой необходимости чтобы совпадали размерности исходного массива и массива назначения.

Нет также никакой необходимости, чтобы оба массива имели строго одинаковое количество рядов и их длину. Когда Вы работаете со строковыми массивами, строки исходного массива "подрезаются" в соответствии с длиной строк массива назначения, если они длиннее или, наоборот, "подбиваются пробелами", если они короче. Числовые массивы обрабатываются так же, за исключением того, что вместо пробелов используются нули.

Предположим, что Вы имеете массив a$(100,30) и он полностью заполнен. Чтобы

добавить к нему еще 20 символьных строк, Вы можете действовать например так:

```
DIM b$(20,30): JOIN b$ TO a$
```

Поскольку мы использовали JOIN, все строки будут реально удалены из массива b$ и помещены в a$, а не просто скопированы, как это было бы при применении COPY. Поскольку никаких параметров вырезки после имени исходного массива не было указано, то все его символьные строки будут перемещены и массив перестанет существовать. При использовании COPY он остался бы нетронутым.

Позиция вставки в массив назначения была указана. По умолчанию будет принято, что это позиция номер 101, то есть следующая за последней. Первая символьная строка из массива b$ станет сто первой в массиве a$, а последняя станет строкой номер 120.

Т.к. в БЕТА-БЕЙСИКе массивы могут часто менять свой размер, то Вам может потребоваться неоднократное применение функции LENGTH чтобы определить размер массива. В вышеприведенном примере LENGTH (1,"a$") дало бы величину 120.

Если Вы примените DIM b$(20,3) или DIM b$ (20,50), а затем сделаете JOIN b$ TO a$, то в результате строки массива b$ будут "подбиты" пробелами или, наоборот "подрезаны" так, чтобы они укладывались в строки массива a$, имеющие 30-символьную длину.

Если у Вас уже есть готовый массив и Вы решите, что в нем надо сделать строки подлиннее, Вы можете это сделать, задав через DIM массив с желаемой длиной только с одним элементом, а потом переместить (JOIN) свой массив в него.

```
DIM b$(1,40): JOIN a$ TO b$
```

Единственный недостаток, к сожалению, состоит в том, что теперь все наши данные будут находиться в b$, а a$ не будет существовать. Чтобы восстановить его, можно сделать обратный ход:

```
DIM a$(1,40): JOIN b$ TO a$
```

До сих пор мы имели дело с полными массивами, но и JOIN и COPY могут работать много гибче. Вы можете указать какие символьные строки, или какие ряды исходного числового массива Вы хотите использовать с помощью выделяющих параметров, а также позицию, в которую произойдет вставка.

Следующие примеры написаны для JOIN, но соответственно могут работать и с COPY.

```
JOIN a$ TO b$(4)
```

- введет весь массив a$ в b$ таким образом, что первая символьная строка будет вставлена после четвертой символьной строки в увеличенном массиве b$.

```
JOIN a$(2 TO 5) TO b$(1)
```

- переместит 4 символьных строки из a$ в начало массива b$. После этого массив a$ станет на четыре строки короче, чем был, а массив b$ станет на четыре строки длиннее.

```
JOIN a$(10) то b$
```

- переместит десятую символьную строку из a$ в конец b$.

с) числовые массивы.

Одномерные и двумерные числовые массивы обрабатываются способом, похожим на символьные массивы. После имени массива должны обязательно стоять скобки, чтобы отличать массивы от простых переменных.

Нижеприведенный пример создает два массива, а затем объединяет их между собой.

```
10 DIM a(8)
20 FOR n=1 TO 8
30   LET a(n)=n
40 NEXT n
50 DIM b(5)
60 FOR n=1 TO 5
70   LET b(n)=n*10
80 NEXT n
90 JOIN b() TO a()
100 FOR n=1 TO LENGTH (1,"a(")
110   PRINT a(n)
120 NEXT n
130 REM печать 1 2 3 4 5 6 7 8 10 20 30 40 50
140 REM b() - не существует
```

28. KEYIN строковая переменная

Клавиша: SHIFT + 4

KEYIN вводит строковую переменную так, как если бы Вы впечатали ее с клавиатуры. Таким образом, есть возможность сделать программу самосоздающейся, хотя это и выходит за пределы, рассматриваемые в данной инструкции. Можно рассмотреть пример автоматического создания строк DATA.

```
10 LET a$ = "100 DATA"
20 FOR n=0 TO 9
30 LET a$ = a$+STR$(PEEK N) + ","
40 NEXT n
50 LET a$ = a$ (1 TO LEN a$ -1): REM удаление последней запятой
60 KEYIN a$
```

После запуска данного фрагмента Вы увидите, что к программе добавилась еще одна строка.

В режимах KEYWORDS 3 и 4 ключевые слова, набранные по символам, будут преобразованы в односимвольные токены.

29. KEYWORDS число.

Клавиша: 8

Оператор KEYWORDS управляет тем, как вводятся и как изображаются на экране ключевые слова стандартного БЕЙСИКА, БЕТА-БЕЙСИКа и символов графики пользователя (UDG).

KEYWORDS 0.

В этом режиме за клавишами в графическом режиме (курсор G) закреплены символы графики пользователя, как в стандартном БЕЙСИКе Вашего "Спектрума".

KEYWORDS 1.

В этом режиме за клавишами в графическом режиме закреплены ключевые слова БЕТА-БЕЙСИКА.

В исходном состоянии после загрузки система находится в режиме KEYWORDS 1. А если Вам надо воспользоваться символами графики пользователя, дайте команду KEYWORDS 0. Выбранный Вами режим KEYWORDS 1 или KEYWORDS 0 не влияет на то, как вводятся ключевые слова - по буквам или как токены, целиком, т.е. выбор режима 0 или 1 не влияет на установки режимов KEYWORDS 2, 3 и 4. После загрузки компьютер находится в состоянии KEYWORDS 3. Текущий режим ввода сохраняется вместе с Вашей программой, если Вы отгружаете вместе с ней код (CODE) самого БЕТА-БЕЙСИКА.

KEYWORDS 2.

В этом режиме все ключевые слова вводятся одним нажатием клавиш, при необходимости с одновременным нажатием шифтов. Ключевые слова стандартного БЕЙСИКа берутся, как обычно. Ключевые слова БЕТА-БЕЙСИКа вводятся в графическом режиме (курсор G), а функции БЕТА-БЕЙСИКа вводятся в порядке FN, буква, знак "$" или "(".

KEYWORDS 3.

Этот режим - тот же, что и KEYWORDS 2, за исключением того, что вводимая строка, прежде чем пойти в память компьютера, сначала проверяется на наличие в ней набранных по буквам ключевых слов и, если они найдены, происходит их замена на токены ключевых слов. По-видимому, это самый удобный режим, т.к. в нем можно работать и с вводом ключевых слов одним нажатием клавиш и со вводом их по буквам. Если нужно выйти из курсора "K", Вы можете использовать ведущий пробел.

KEYWORDS 4.

В этом режиме нет курсора "K", т.е. все ключевые слова вводятся только по буквам. Форсировать появление курсора "K", тем не менее, все же возможно. Это делается нажатием клавиш SYMBOL SHIFT и ENTER.

Этот режим, возможно, предпочтут те, кому может потребоваться совместимость с другими моделями персональных компьютеров.

d) Распознавание ключевых слов.

Есть небольшое ограничение в режимах 3 и 4, которое заключается в том, что Вы не можете использовать имена переменных и процедур, совпадающие по написанию с ключевыми словами. Поскольку при запоминании программной строки в памяти компьютера может произойти их замена на токен ключевого слова. Тем не менее, ключевое слово может быть частью имени переменной или процедуры без проблем.

Ключевые слова в этих режимах распознаются как набранные прописными, так и строчными буквами. Мы Вам рекомендуем использовать строчные буквы, тогда после конверсии этих ключевых слов в токены, Вы сможете наглядно увидеть, какие ключевые слова были преобразованы и, возможно, найдете синтаксическую ошибку.

Символы, стоящие непосредственно перед ключевым словом или за ним не могут быть буквами или символами подчеркивания "_".

Printa - не будет конвертировано в PRINT a, т.к. компьютер предположит, что это имя переменной или процедуры.

Print a - будет преобразовано в PRINT a. Ниже приведены несколько примеров возможных строк и показано их возможное преобразование в результате "токенизации".

```
print 10......................PRINT 10
print fork, total.............PRINT fork, total
alter to ink3, paper1.........ALTER TO INK 3, PAPER 1
print string$(10,"Plot")......PRINT STRING$(10,"Plot")
goto10........................GO TO 10
go to x.......................GO TO x
gotox....... .................gotox
defproc pink..................DEF PROC pink
mat_print.....................mat_print
```

Пример с GO TO показывает, что внутренние пробелы в этот оператор можно и не включать. Точно так же и "gosub", "onerror", "defproc" будут распознаны, как полноценные ключевые слова.

Слово "ink", входящее как составная часть в "pink" и "print", входящее в "mat_print" распознаны не будут, т.к. перед ключевым словом не должно быть буквы или знака "_".

30. LET переменная = число <, переменная = число>...

Маленькое усовершенствование старой команды LET позволяет выполнять серию присвоений, разделяя их запятыми. Это сокращает объем занятой памяти (по одному байту на каждый опущенный LET), делает ввод более удобным и листинг более читаемым. Так, запись

```
10 LET x=1,y=2,z=3,a$="y",b$="n"
```

может заменить:

```
10 LET x=1: LET y=2: LET z=3: LET a$="y": LET b$="n"
```

31. LIST <номер строки> TO <номер строки>

или LLIST <номер строки> TO <номер строки>

Вы видите по синтаксису, что здесь есть небольшое добавление к стандартному БЕЙСИКу. Вы можете выводить на экран или принтер заданный Вами блок программы. Если первый номер строки не указан, то по умолчанию принимается строка, следующая за нулевой. Если второй номер строки не указан, предполагается по умолчанию последняя строка программы. Если оба параметра опущены, Вы получаете эквивалент обычной команде LIST.

```
LIST 20 TO 100
LIST TO 200
LLIST 100 TO 180
```

Если строка с первым номером существует, то при листинге она будет изображена с курсором ">", т.е. она готова к вызову на редактирование.

Если оба номера совпадают, то будет изображена только одна строка.

32. LIST DATA

или LIST VAL
или LIST VAL$
Эти разновидности команды LIST позволяют распечатать сводку переменных:
LIST DATA - все переменные
LIST VAL - числовые переменные
LIST VAL$ - строковые переменные.

"Спектрум" имеет 6 типов переменных. Команда LIST VAL распечатывает из них 4 типа в следующем порядке:

1. Числовые массивы.
2. Переменные циклов FOR-NEXT.
3. Переменные с односимвольными именами.
4. Переменные с многосимвольными именами.

Команда LIST VAL$ распечатает остальные два типа переменных

5. Символьные массивы.
6. Обычные символьные переменные.

Команда LIST DATA распечатает все 6 типов переменных.

Переменные каждого типа распечатываются в алфавитном порядке (для переменных с многосимвольными именем в расчет принимается только первая буква). Пример того что может дать LIST DATA приведен ниже:

```
d(10,4)
k(3,3,4)
n STEP 1    500         LN 200
g           3.5
j           100
s           23.1

applen      1
number      9999
xos         0
xrg         256
yos         0
yrg         176

t$(100,10)

a$ LEN 5    "Hello"
b$ LEN 40   "Too long too..."
e$ LEN 5    "Bang!"
```

Для массивов изображается только их размерность, но не содержание. Переменные циклов FOR-NEXT можно отличить от прочих благодаря присутствию параметра STEP и параметра LN (looping number - номер строки, из которой производится возврат в голову цикла). Для длинных строковых переменных изображаются только первые 15 символов.

33. LIST DEF KEY

Здесь DEF KEY располагается на клавише SHIFT + 1.

См. также DEF KEY.

Эта команда позволит распечатать список клавиш, определенных пользователем. Сначала распечатывается клавиша, а затем символьная строка или оператор, которые за ней закреплены. Если в назначении клавиши последним символом является двоеточие, то полагается, что при нажатии этой клавиши то, что за ней закреплено, сопровождается последующим ENTER, т.е. сразу после нажатия содержимое появляется в нижней части экрана в системном окне.

Михайленко В.С.

ЗАЩИТА ПРОГРАММ

Продолжение. (Начало см. стр. 9-16, 53-60)

Глава 4. Прочие приемы защиты.

4.1 Запуск программ в кодах.

О том, как запустить программу в машинных кодах, наверное, знают все пользователи "ZX SPECTRUM". Эта информация изложена во всех справочниках по данному типу компьютеров.

Вкратце напомним основные положения данной системы команд.

Для вызова подпрограмм в машинных кодах используется функция USR, составленная из ключевых слов английского языка:

```
User SubRoutine
```

В компьютерах типа ZX SPECTRUM эта функция может использоваться двояко. Во-первых, она применяется в случаях, когда необходимо вызвать подпрограмму, написанную машинными кодами и расположенную в памяти по известному адресу.

Она также может применяться для записи данных графики, устанавливаемой пользователем, в зарезервированное место в конце памяти.

Нас интересует случай применения USR для вызова подпрограмм в машинных кодах. Для запуска данной подпрограммы USR используется с ключевым словом Бейсика RANDOMIZE или PRINT. После комбинации этих слов указывается цифровая величина, например:

```
90 RANDOMIZE USR 30000
100 PRINT USR 45000
```

Если вместо определенного цифрового значения используется выражение, то оно должно быть заключено в скобки. Указанная величина округляется до ближайшего целого числа - это адрес памяти, с которого и должна стартовать записанная машинными кодами подпрограмма. В общем случае адрес начала подпрограммы и адрес, с которого она стартует, могут не совпадать, хотя очень часто они совпадают.

Результат функции USR - значение, находящееся в паре регистров ВС микропроцессора. Комбинация ключевых слов RANDOMIZE USR или RESTORE USR только запускает подпрограмму, тогда как PRINT USR еще и индицирует содержимое пары регистров ВС на экране.

Фактически, это достаточно широко известная информация и на ней не стоило бы останавливаться, если бы не одно но:

- современные системы защиты, используя встроенные функции компьютера, создали новые системы команд для запуска подпрограмм в машинных кодах.

То, что из этого следует, очевидно. Засекретив начало Вашей подпрограммы в кодах, Вы сбиваете с толку "хакеров", а это в свою очередь защищает Вашу программу от несанкционированного просмотра. Рассмотрим более подробно эти новые ухищрения.

Выше были рассмотрены команды, достаточно широко известные и наиболее часто применяемые. Чуть реже встречается комбинация

```
LET A = USR 30000
```

Фактически эта команда полностью аналогична рассмотренным выше - она запускает подпрограмму в кодах с адреса, указанного в функции USR - в данном случае, с адреса 30000. Любопытно, что эта комбинация достаточно широко используется при работе со

встроенным калькулятором "СПЕКТРУМа". (Для тех, кто интересуется этим более подробно, рекомендуем изучить трехтомник "ИНФОРКОМа" по программированию в машинных кодах, где это применение описано более подробно).

Но, если читатель мог встретить подобную комбинацию в некоторых программах, то описанные ниже выражения встречаются достаточно редко. Я имею ввиду применение для запуска подпрограмм в кодах некоторые функции Бейсика.

В частности, если Вы подадите команду:

```
GO TO USR 30000
```

то выполнение команды осуществится аналогично

```
RANDOMIZE USR 30000
```

Среди таких модификаций команды RANDOMIZE USR следует отметить еще ряд команд, в частности:

```
LIST USR
```

и

```
CLOSE# USR
```

Однако следует учитывать, что применение данной системы команд будет иметь весьма незначительный эффект, если не учитывать одного нюанса, а именно: несмотря на вариации первого ключевого слова, наличие команды запуска подпрограммы в кодах достаточно очевидно. Вот почему эту комбинацию следует использовать совместно с другими приемами.

Для пояснения вышеизложенного рассмотрим применение подобной системы команд у Billa Gilbert-а (следует отметить, что он берет на вооружение все новые системы защиты и поэтому изучение его методов и приемов значительно повысит Ваш профессиональный уровень. В самом деле, он взламывает программу и должен поставить на нее такую защиту, чтобы другому вскрыть ее уже не удалось. Причем следует отметить, что он использует и достаточно оригинальные методы, те свои собственные разработки, которые не встречаются ни в каких других программах).

Когда мы просматривали дампинг одной из программ, вскрытых Биллом Гилбертом, то обнаружили достаточно любопытную комбинацию символов. После номера строки следовало:

```
CLOSE # USR 0
```

и т.д.

Сначала мы приняли эту последовательность символов за начало программы в машинных кодах, размещенной в Бейсике. В самом деле, эта последовательность не несла в себе никаких информационных признаков о командах Бейсика. Но в то же время, если бы это было начало подпрограммы в кодах, то первым должен был бы следовать символ REM, поскольку именно после него размещается подобного рода код. Это заставило нас более внимательно проанализировать данную строку и секрет был быстро разгадан.

Естественно, эта комбинация свидетельствовала о начале подпрограммы в машинных кодах, но не думайте, что она была равнозначна команде

```
RANDOMIZE USR 0
```

Ноль в данном случае был ширмой, за которой скрывалось истинное значение числа, указывающего адрес начала программы в машинных кодах. Этот прием с подменой значений достаточно подробно описан нами в главе 11 данного пособия.

4.2. Защита, основанная на использовании вариаций числа PI вместо цифровых констант.

В главе I были описаны специальные POKE которые применяются в различных целях в том числе и для защиты от остановки командой BREAK во время работы Вашей программы. Таких методов существует несколько, но общие критерии их применения аналогичны - Вам необходимо изменить значение одной из системных переменных и тогда после нажатия BREAK (если Вы ввели это изменение до нажатия данной клавиши), у Вас не произойдет остановки программы, а произойдет сбой в работе компьютера и машина либо зависнет, либо произойдет сброс системы, аналогичный нажатию кнопки RESET.

Но с тем, чтобы скрыть от начинающего "хаккера", какую именно системную переменную Вы меняете и какое значение Вы туда засылаете, можно использовать следующий прием.

В частности, Вам необходимо изменить содержимое ячейки 30000, сделав его равным 150.

Традиционно данная команда выглядела бы следующим образом:

```
10 POKE 30000, 150
```

Но ваша задача запутать "хаккера". Вместо чисел желательно использование каких-либо переменных, причем таким образом, чтобы они не описывалась в этой же программе оператором LET.

Поясним вышеизложенное на примере, чтобы заменить адрес традиционным способом, нам пришлось бы вводить еще одну строку программы

```
10 LET A1 = 30000
20 POKE A1,150
```

Но этого можно избежать, если задать переменную A, подав команду с клавиатуры:

```
LET A = 30000
```

После этого значение переменной A будет занесено в память компьютера и мы сможем вызывать его по мере необходимости. Одно условие - это значение будет уничтожено, если Вы подадите команду RUN, т.к. эта команда полностью освежает буфер переменных. Для запуска Вашей программы необходимо использовать функцию GO TO. Следует отметить один нюанс - автозапуск программы по команде LINE N (со строки N) осуществляется функцией GO TO N, поэтому Ваше значение будет сохранено в памяти компьютера.

Другой аспект проблемы - это использование вместо числа 150 его кодового слова среди символов символьного набора компьютера. В частности, символу 150 соответствует графический код "G". Следовательно, если Вы используете команды:

```
LET A = 30000
10 POKE A, CODE "G"
```

- (используется символ графики), то это будет полностью аналогично первой строке программы в этой статье. Кроме функции CODE можно использовать и производные функции PI:

NOT PI - равноценно 0
SGN PI - равноценно 1
INT PI - равноценно 3

Значения, которые не находят эквивалента в производных PI могут быть образованы с использованием различных комбинаций, в частности:

```
6 = INT PI + INT PI
```

В заключение этой статьи натолкнем читателя на очень любопытную мысль. В частности, раз Вашей задачей является как можно более запутать "хаккера", то почему бы Вам не набрать вместо имени переменной (в нашем примере используется переменная A) имя, аналогичное одному из ключевых слов Бейсика. Смею Вас заверить, это будет иметь поразительный эффект. В частности, постарайтесь представить себе неопытного "хаккера", увидевшего на экране следующую комбинацию:

```
LET USR = 30000
10 POKE USR, SGN PI
```

Неправда ли, смотрится впечатляюще для тех, кто не знает, что здесь USR не ключевое слово, а имя переменной и набирается по буквам.

4.3 Ключевые слова Бейсика в "хэдере".

Одним из наиболее любопытных приемов защиты, шокирующим начинающих пользователей "СПЕКТРУМа", является использование управляющих кодов в хэдере. Но не менее интересным приемом является использование в хэдере ключевых слов Бейсика. Эта тенденция просматривается во многих программам и позволяет сэкономить и без того

скудное пространство хэдера с целью создания полноценного названия.

Поясним вышеизложенное на примере.

Тем, кто внимательно ознакомился с главой II данного пособия, наверняка известен тот факт, что в хэдере содержится всего 10 символов, отведенных под имя программы. При использовании управляющих кодов количество свободных символов сокращается. Однако, если Вам хочется иметь полноценную заставку - заголовок, то одним из методов, позволяющим совместить это желание с желанием использования управляющих кодов, является применение в хэдере ключевых слов Бейсика.

Естественно, ключевые слова не могут в полном объеме характеризовать Ваше название и вряд ли смогут передать всю гамму названий, набираемых отдельно по буквам. Но, в то же время, их применение создаст эффект высокой защищенности программы, а разного рода финты достаточно сильно действуют на непрофессионалов отталкивающим образом.

Рассмотрим любопытный пример - название программы "GAME OVER". Его можно было бы написать и по отдельной букве, но тогда это не дало бы возможности использовать в хэдере управляющие коды, которые в моем варианте уничтожают после загрузки на экране слово PROGRAM. Для экономии места здесь используется оператор Бейсика OVER. Среди остальных, известных мне названий программных файлов, приведу наиболее распространенные, использующие ключевые слова Бейсика:

"MVS FORMAT"
"MDS FORMAT"
"B.G. FORMAT" - используется Биллом Гилбертом.

Конечно, не все Ваши названия можно выразить подобным образом, но, в то же время, следует отметить, что можно использовать слова-синонимы, аналогичные ключевым словам Бейсика. В любом случае, возможность такой замены Вам следует иметь в виду.

Следует отметить, что использование ключевых слов Бейсика в хэдере несколько лет назад еще использовалось для защиты программ от копирования. Когда копировщик считывал название программы, то он не мог распознать символ ключевого слова Бейсика и это вызывало сбой в его работе. Однако, последние модели копировщиков уже избавлены от этого недостатка.

4.4 Мерцающий заголовок.

Многие из читателей наверняка пользовались программой COPY-COPY. И, вероятно, обращали внимание на тот факт, как интересно там организован хэдер.

После загрузки хэдера на экране начинают попеременно мигать символы COPY-COPY. Подобная комбинация не имеет принципиального значения для непосредственной защиты Бейсик-программ, но имеет косвенный эффект.

Организация защиты подобных программ воспринимается как очень хорошая, особенно среди начинающих "хаккеров". Этот косвенный эффект совместно с действительно хорошей защитой, основанной на приемах, описанных в этой работе, превратят Вашу программу в "неприступную" крепость для взломщиков.

Итак, вкратце рассмотрим, как работает данный хэдер. Для тех,[1]

В программе COPY-COPY символы в заголовке имеют следующую последовательность :

[1] В оригинале пропущена строка (Прим. OCR)

```
18        - управление FLASH 1
255 COPY
6         - PRINT запятая 20
1         - управление INVERSE
255 COPY
0
8
```

В данном случае набор этих символов следует рассматривать, как определенную программу, предписывавшую компьютеру определенные действия. Рассмотрим, какие команды будет выполнять компьютер в данном случае.

Теоретически наш хэдер должен был бы появиться на экране (имеется в виду название программного файла) в виде печатных символов, но в данном случае здесь нет таковых. Первая пара символов 18 и 1 - это под управление FLASH и соответственно значение, устанавливающее режим мерцания. Далее следует символ COPY, который распечатывается на экране и начинает мерцать. После этого следует управляющий код 6 - т.н. PRINT-запятая, которая перемещает курсор в следующую половину экрана, аналогично запятой, использующейся в операторе PRINT. Следующая пара символов устраивает значение INVERSE - в данном случае включает инверсный режим. Теперь мерцание второго символа "COPY" будет находиться в противофазе с мерцанием первого, что и создает эффект попеременного мерцания.

Чтобы создать такой хэдер на компьютере, Вам необходимо создать строку выгрузки в Вашей программе. В ней должны находиться операторы SAVE "название" и т.п. Причем вместо управляющих кодов должны стоять пробелы. После этого Вам надо получить дамп памяти по одной из предложенных в главе 2 программ и определить местонахождение заголовка. После этого Вы засылаете в эту область памяти последовательность цифр, которая приведена выше. Естественно, приноровившись, Вы сможете создавать и более оригинальные комбинации, в частности, например, попробуете убрать слово PROGRAM или разместить заголовок в другой точке экрана. Для этого Вам необходимо внимательно изучить раздел "Управляющие коды" главы II данной работы.

Поскольку подобные коррективы в хэдере не имеют принципиального значения, мы не будем останавливаться на заголовке другого цвета и в другом месте экрана (отличном от привычного). Надеемся, что читатель сам сможет сделать это.

4.5 Метод защиты, основанный на смещении программ в кодах при попытке взлома программ.

Если читатель внимательно изучил все материалы, изложенные выше, то он приобрел достаточно высокий уровень подготовки для защиты своих программ. Тем не менее, поработав с ними длительное время, он начинает осознавать, что многие из приемов, применяемых им, достаточно широко известны в сфере программистов для "ZX SPECTRUM".

Возникает необходимость создания новых приемов защиты, никем до этого не применявшихся, чтобы обеспечить надежность защиты собственных программ. К этой цели можно идти двояко:

1) Создавая свои сугубо оригинальные приемы;

2) Создавая новые комбинации из уже известных приемов, улучшающие качество и надежность систем защиты;

Первый путь, безусловно, более надежен, но он требует виртуозного знания всех тайников компьютера, что требует, в свою очередь, достаточно целенаправленной работы с компьютером в течение длительного времени. Большинство же читателей вряд ли обладают возможностью длительно и целенаправленно изучать компьютер и поэтому мы предлагаем им избрать второй подход. Если первый подход хорош только для опытного, квалифицированного специалиста, то второй может подойти любому пользователю, внимательно изучившему данную статью. Второй метод включает в себя комбинирование изложенных здесь приемов с целью повышения эффективности защиты. На первый взгляд выгода от такого подхода не столь очевидна, однако не стоит забывать, что порой

интересная комбинация уже известных методов может иметь любопытный косвенный эффект. В качестве примера рассмотрим комбинацию из известных Вам методов совмещения в Бейсик-строке машинных кодов и метода защиты от листинга с использованием управляющих кодов.

Когда мы с Вами рассматривали совмещение с Бейсиком машинных кодов, то было отмечено, что удобнее всего для Вас использовать первую строку программы для непосредственного совмещения, что связано с правильным нахождением адреса начала загрузки.

Безусловно для Вас такой метод является самым удобным, но не менее удобным он является и для взломщика. Ведь фактически в такого рода программах ему даже не приходится искать адрес начала подпрограммы в кодах - первая строка Бейсика начинается с фиксированного адреса оперативной памяти, на который указывает системная переменная PROG и без подключенной периферии и умышленных изменений значение, находящееся в этой паре байтов указывает на адрес 23755. Соответственно, для нахождения адреса размещения такой подпрограммы достаточно к этому значению прибавить 5 (эти 5 байтов включают в себя номер и длину строки - по два байта на каждый параметр соответственно - и код оператора REM). Кроме этого значение адреса начала подпрограммы можно узнать из значения, стоящего после RANDOMIZE USR или его вариаций.

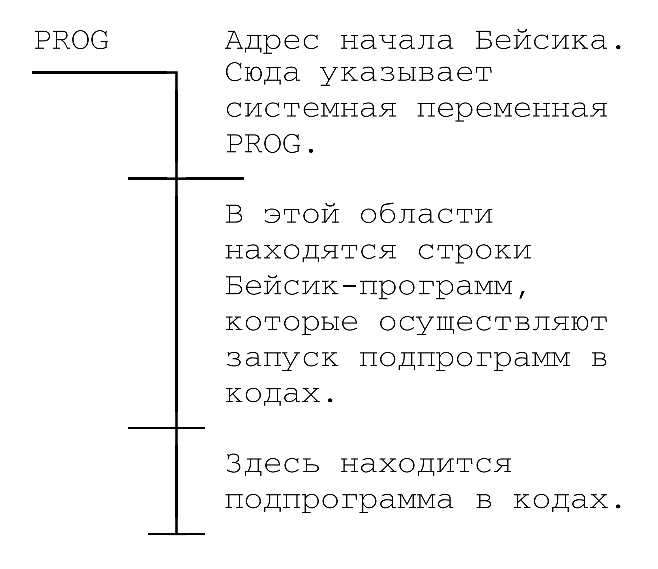
Одним из выходов является создание "плавающей" подпрограммы в кодах и задание "фальшивого" адреса старта данной подпрограммы.

Рассмотрим более подробно каждый из методов.

Как Вам уже вероятно известно, Бейсик-интерпретатор размещает строки программ последовательно. Т.е. если у Вас есть строки с номерами 5 и 10 и Вы хотите ввести строку с номером 7, то интерпретатор Бейсика разместит эту строку между строками 5 и 10. Когда он будет делать это, то строка с номером 10 переместится в более старшие адреса области ОЗУ, причем ее старое местоположение будет отличаться от нового на длину строки 7.

Теперь представьте, что строка с номером 10 - это строка, в которой расположена наша подпрограмма в кодах, а строки 5 и 7 на начальном этапе отсутствуют. Таким образом, после формирования данной строки с машинными кодами после оператора REM нам необходимо сместить ее в верхние адреса ОЗУ. Для этого все команды управления на Бейсике размещаем в строках, находящихся выше строки с подпрограммой в машинных кодах. (Для того, чтобы найти адрес начала подпрограммы в кодах, Вам придется использовать метод распечатки дампинга памяти, который будет подробно рассмотрен в конце этой статьи).

Если Вы начнете придерживаться изложенных выше рекомендаций, то можно сказать, что первый недостаток нам удалось преодолеть. В самом деле, теперь у Вас нет фиксированного адреса, с которого начиналась бы подпрограмма в кодах и этот адрес зависит от длины строк Бейсика, в которых бы осуществлялся запуск данной программы.



Вторая часть нашей программы будет создаваться временно, для получения дампинга и будет располагаться после подпрограммы в кодах.

Теперь перейден ко второй части проблемы. Нам необходимо скрыть адрес начала подпрограммы в кодах. Здесь мы используем комбинирование уже известных методов для создания очень интересного эффекта.

В целях более полной защиты, применим двойную защиту, которая при правильном использовании превратится в тройную защиту. Для этого нам придется, во-первых, использовать управляющие коды для слияния цветов INK и PAPER с целью сокрытия информации, а во-вторых, создать фальшивый адрес старта подпрограммы в кодах с использованием управляющего кода 14. Оба эти метода подробно описаны в главе 2 и поэтому Вам необходимо подробно ознакомиться с ними. Ниже будет приведен лишь алгоритм создания защиты и поэтому, если Вы не уяснили себе информацию, изложенную там, Вам будет достаточно сложно сориентироваться в происходящих процессах.

Итак, для начала Вам необходимо полностью оформить аппарат Бейсика, т.е. создать строки программы, которые бы выполняли необходимую Вам операцию. Естественно, все они должны иметь номер меньший, чем строка, в которой находится подпрограмма в кодах. Среди команд, помещенных в этих строках, обязательно должна быть команда запуска подпрограммы в кодах, причем номер, на который указывает RANDOMIZE USR (выгоднее применять разновидность этой команды - раздел 4.1) может быть произвольным. Обязательным условием является лишь то, чтобы он содержал не менее 5 знаков (и не более), т.е. например, запуск осуществлялся бы командой RANDOMIZE USR 25000. Это связано с тем, что при замене этого номера на истинный, который мы будем находить в следующей операции нашего алгоритма, программа в кодах может сместиться, из-за разного количества символов в номере.

После того, как Вы это сделали, нам необходимо будет зарезервировать место под управляющие коды, используя оператор REM. Всего у нас будет использоваться 4 байта INK CONTROL, значение цвета INK, PAPER CONTR и значение цвета PAPER.

Для этого в первой идущей в Вашей программе строке необходимо установить REM и три произвольных символа. Оператор REM устанавливается перед первым идущим в строке оператором.

После того, как мы установим точное расположение подпрограммы в кодах, мы заменим эти зарезервированные 4 байта на управляющие коды.

Следующим шагом мы как раз и установим точное местонахождение программы. Для этого в самом конце Бейсика разместим программу распечатки дампа памяти.

```
9997 FOR I = 23755 TO 64000
9996 PRINT I; TAB 7; PEEK I; TAB 12; CHR PEEK I
9999 NEXT I
```

Сразу же следует оговориться, что программа может останавливаться по ошибке

INVALID COLOR - неправильный цвет

NUMBER TOO BIG - большое целое число и т.п.

В этом случае Вам необходимо подать команду с клавиатуры:

```
NEXT I
```

и распечатка продолжится.

После того, как Вы стартуете данную программу командой GOTO 9997, Вы получите распечатку в следующем формате:

Номер ячейки памяти	десятичное значение содержащееся там	символьное значение содержащееся в этой ячейке памяти

Внимательно ее анализируя, Вы более подробно изучите структуру своей Бейсик-строки. Просматривая таким образом содержание памяти в области Вашей Бейсик-программы, Вам необходимо найти и записать номера следующих ячеек памяти:

1) Номер ячейки, где находится оператор REM, после которого следуют 4

зарезервированных под управляющие коды байта.

2) Номер ячейки, в которой содержится первая цифра числа, находящегося после RANDOMIZE USR.

3) Адрес ячейки, с которой начинается Ваша подпрограмма в кодах.

После того, как эта информация Вами получена, необходимо занести в строку, запускающую Вашу программу и содержащую RANDOMIZE USR, правильный адрес расположения подпрограммы в кодах, полученный в пункте 3.

Когда это Вами сделано, Вам необходимо это число "фальсифицировать".

Как Вам уже вероятно известно, в "ZX SPECTRUM" в памяти числа хранятся дважды - первый раз значение, которое печатается на экране, а второй раз - в пятибайтном формате, расположенное после кода 14. В данном случае нам необходимо изменить именно первое значение, которое будет распечатываться на экране.

Первый байт этого значения расположен по адресу, который мы получили в пункте 2. Вы можете убедиться в этом снова, запустив программу дампинга и увидев в этом же месте новое значение числа. Для того, чтобы изменить это значение, Вам необходимо заслать в эти ячейки памяти значения ASCII кодов "фальшивого" числа с помощью оператора POKE.

Следующим пунктом нам необходимо скрыть листинг данной программы. Для этого сделаем одинаковым цвет чернил и бумаги с помощью управляющих кодов. Например, мы хотим, чтобы это был белый цвет. Тогда нам необходимо последовательно заслать в ячейки памяти, адреса которых мы получили в пункте 1, следующие значения:

16 - изменение INK
7
17 - изменение PAPER
7

После того, как Вы выполните это командой POKE, на экране уже невозможно будет получить листинг.

Теперь Ваша программа защищена от просмотра командой LIST. Осталось удалить последние строки, осуществлявшие дампинг памяти. Сделайте это, подав команды

```
9997 ENTER
9998 ENTER
9999 ENTER
```

Теперь рассмотрим, какие новые качества получила наша защита после такой комбинации.

Если мы вызовем первую строку программы, в которой заложены управляющие коды, для редактирования и уничтожим эти коды командой DELETE, то теоретически мы увидим адрес начала программы в кодах - практически же Вы знаете, что это значение будет неправильным. Но, даже если Вам и удастся определить правильный адрес старта данной подпрограммы в кодах, просмотрев через PRINT PEEK то, что стоит после кода 14, Вам все равно не удастся ее правильно дисассемблировать, поскольку Вы уничтожили управляющие коды командой DELETE, то есть уменьшили размер первой строки на 4 байта и, тем самым, сместили адрес начала данной подпрограммы.

Таким образом, мы видим, что совмещение хорошо известных Вам приемов может приводить к совершенно новым результатам и порождать любопытные побочные эффекты, которые в данном случае тоже используются для защиты Вашей программы.

* * *

Том 2. "Техника взлома компьютерных программ ZX-SPECTRUM. "

Глава 1. Введение

Компьютеры "ZX-SPECTRUM" были разработаны в Англии и поэтому многие пользователи в нашей стране сталкиваются с трудностями освоения программ. Это касается не только проблем языкового барьера. Трудности, связанные со слабым знанием английского языка в большинстве случаев решаются за счет его изучения самим пользователем, потому что без определенного уровня достаточно проблематично осваивать интересное программное обеспечение. Но, кроме барьера языкового, существует и барьер программной защиты. Ведь многие пользователи, проработав определенное время с программой, желают залезть к ней в "нутро" и осуществить какие-либо изменения. Это могут быть изменения, направленные на русификацию программы, изменения, направленные на адаптацию программы под требования пользователя или же просто просмотр с целью изучения приемов программирования. Кроме этого, существует проблема с копированием некоторых защищенных программ, копирование которых невозможно осуществить без определенного уровня подготовки, а также исследования конкретной структуры защиты данной программы.

В преодолении этих трудностей Вам поможет изучение техники взлома защищенных программ, изложение которой и преследует своей целью данное пособие. Взлом компьютерных программ занятие достаточно увлекательное и, в то же время, многие используют это занятие в неблаговидных целях. В частности, очень многие взломщики оставляют после себя определенные надписи, т.е. используют взлом программ для саморекламы, но, как Вы скоро поймете, это не является самым интересным. Куда более любопытным нам представляется использование взлома компьютерных программ для повышения Вашей техники программирования, а также более детального изучения имеющегося у Вас компьютера, превращения его из обыкновенной игрушки в серьезный инструмент для решения Ваших задач.

Общие рекомендации.

Если Вы внимательно изучили т. 1, то теперь Вам известны основные принципы защиты компьютерных программ, и, несмотря на то, что техника защиты постоянно совершенствуется, все, что создается отвечает известному Вам принципу: "Все новое - это хорошо забытое старое". А это значит, что все новые приемы защиты являются модернизацией уже известных.

Но это происходит не всегда. Иногда появляются исключительно новые приемы, которые нельзя классифицировать ни по какой старой схеме. К счастью, это скорее исключение, чем правило. А это говорит о том, что Вам следует рассматривать защиту новой программы как набор уже известных Вам приемов или же их комбинацию, пока не станет ясно, что это что-то новое.

Второе, о чем Вам необходимо серьезно задуматься, так это о том, с какой целью Вы взламываете программу. Наиболее распространенным, на наш взгляд, является случай, когда Вы желаете изучить новые приемы программирования или же адаптировать программу для себя. Эти случаи предусматривают вариант, когда Вам придется досконально изучить содержание программы, чтобы понять принцип ее работы и внести необходимые изменения.

После того, как Вы определились в цели взлома, Вам необходимо подумать о том, как наиболее рационально загрузить программу, чтобы наиболее быстрым способом обезвредить защиту.

Для того, чтобы иметь хоть маломальское представление о типе защиты, примененном в данной программе, Вам необходимо загрузить ее и нажать клавишу BREAK (это необходимо узнать для того, чтобы впоследствии ликвидировать защиту от BREAK одним из предложенных далее способов) и если программа не "зависла", то испробовать, что получается при нажатии клавиши LIST.

Итак, Вы определились в целях взлома и осуществили предварительную подготовку. Мы хотим искренне надеяться, что Ваши цели будут такими, какими представляем их мы или аналогичными.

Структура фирменной программы.

Когда Вы загружаете какую-либо игровую программу, то перед тем, как пойдет загрузка, Вы набираете с клавиатуры команду LOAD "".

Подобная комбинация говорит о том, что первым будет загружаться Бейсик-файл, а уж дальнейшая загрузка пойдет в соответствии с командами данного Бейсик-файла. Рассмотрим более подробно, как происходит загрузка этого Бейсик-файла (фактически данная информация относится к загрузке любого файла в компьютере).

В течение первых 1-2 секунд считывания мы видим на экране широкие красные и синие полосы, а также слышим длительный однотонный звук. Это так называемый пилот (или пилоттон, или просто тон), который позволяет компьютеру синхронизироваться с сигналом с ленты, с которой он будет считывать программу, что обеспечивает надежность ввода и вывода кодов в память компьютера. (Как Вам уже вероятно известно, все программы состоят из кодов. Бейсик-программу также можно представить, как последовательность кодов), затем появляются мерцающие тонкие желтые и синие полоски, свидетельствующие о том, что компьютер считывает в память байты информации. Первый раз эти желто-синие полосы мерцают кратковременно, поскольку компьютер считывает в память заголовок. Этот заголовок (так называемый "хэдер") и содержит в себе всю информацию о последующем загружающемся блоке кодов и имеет размер всего 17 байтов. Поскольку у нас считывается Бейсик-файл, то появляется надпись "PROGRAM", но возможно появление и других надписей, в частности:

BYTES - байты (коды);

CHARACTER ARRAY - символьный массив;

NUMBER ARRAY - числовой массив.

После этой надписи будет стоять название программы.

Теперь после небольшого перерыва начнется второй пилоттон, а после него считывается собственно сама программа.

Если мы подавали команду LOAD "", то у нас загружался Бейсик-файл и после загрузки компьютер поступает в соответствии с данными, поступившими из хэдера. Для Бейсик-программы это может быть либо автостарт программы со строки n (если программа была записана на магнитофон командой:

```
SAVE "название " LINE
```

либо остановка в ожидании команды RUN.

В большинстве фирменных программ происходит автостарт и, в соответствии с командами Бейсика, выполнение программы продолжается, что позволяет осуществить дальнейшую загрузку блоков. Но теперь Вам достаточно очевидно, что если взломать Бейсик-файл и внести в него соответствующие изменения, то Вы сможете самостоятельно и целенаправленно отслеживать процесс загрузки и управлять им.

Именно Бейсик-файл наиболее серьезно защищается производителями и именно на нем Вам следует акцентировать свое внимание, поскольку именно там содержится первичная информация о процессе загрузки последующих блоков, изменив которую Вы сможете воздействовать на дальнейший процесс загрузки.

Естественно, следующим этапом нашим совместных исследований должно стать рассмотрение приемов вскрытия данного Бейсик-файла.

Небольшая историческая справка.

Поскольку данный раздел называется "Структура фирменных программ", мы постараемся Вам изложить в общих чертах эту структуру, несмотря на то, что могли только ограничиться информацией о структуре, необходимой для взлома.

Итак, как Вы уже теперь поняли, загружающийся в компьютер файл состоит из двух частей: хэдера и собственно загружающейся после него программы (для удобства будем впоследствии называть эту совокупность блоком кодов). В хэдере содержится вся

информация о загружающемся после него блоке.

Таким образом, структуру первых программ для "ZX SPECTRUM" можно представить, как совокупность Бейсика и машинных кодов. Машинные коды составляли саму программу, в то время как Бейсик осуществлял загрузку этих кодов в память компьютера, после чего осуществлялся их запуск из Бейсика с использованием функции USR. Таким образом, Бейсик состоял из совокупности команд LOAD "" CODE которую завершала функция USR (Автор не включает сюда все разнообразие текстовой информации, которая выводилась на экран с использованием оператора PRINT). Взлом таких программ с целью изучения вообще являлся личным делом. В последовательность команд Бейсика вводится оператор STOP между совокупностью команд LOAD "" CODE и функцией USR, после чего программа в необходимом для Вас месте останавливалась, и Вы могли беспрепятственно исследовать как структуру, так и содержание самой программы. Фактически мы можем сказать, что эти программы не были защищены, что трудно сказать о более поздних фирменных разработках. Несмотря на все разнообразие появившихся типов защит, мы можем выделить два направления защиты в области загрузки. Это загрузка с хэдером и загрузка без хэдера.

Как действует спектрумовская загрузка с хэдером, нам уже известно. Был, правда, вариант защиты с нестандартный хэдером, который отрабатывался специальной программой. Для загрузки программ этот метод не нашел широкого применения и был вскоре практически полностью вытеснен методом безхэдерной загрузки, однако впоследствии программисты вновь обратились к нему, когда для "Спектрума" вышла масса программ с догружающимися уровнями. Именно при догрузке уровней этот метод получил наибольшее распространение.

Второй метод - т. н. загрузка без хэдера применяется практически во всех фирменных программах последних годов выпуска. У него существует множество разновидностей, но фактически все они сводятся к созданию новой системы хранения данных о вводящимся в компьютер блоке кодов, то есть фактически, ту информацию, которую раньше передавал в компьютер хэдер, теперь передает специальная программа в кодах. Разновидностями же этого метода безхздерной защиты являются обращение либо ко встроенной в ПЗУ подпрограммы загрузки, либо обращение к одной из загружаемых в ходе работы программы подпрограмме, так называемой программе-загрузчику последующих блоков. Следует отметить, что применение своей программы-загрузчика преследует несколько целей:

- создание необычных эффектов в ходе загрузки;
- перемещение загруженного блока кодов в другое место памяти, чтобы обеспечить защиту;
- выполнение этих функции одновременно.

Наглядным примером первого типа являются программы MIKKI и KUNG FU второго типа GREEN BERET и третьего - BOMB JACK.

Разумеется, здесь перечислены не все цели, ради которых применяется конкретная программа-загрузчик собственного изготовления. Но в общем случае применение такой программы говорит о методе бесхэдерной загрузки.

Применение загрузки без хэдера становится возможным благодаря знанию встроенных процедур "ZX-SPECTRUM". В частности, чтобы вызвать встроенную подпрограмму загрузки, Вам необходимо внести в регистровые пары HL и BC соответственно адрес, с которого начинается загрузка кодов и их длину, а также соответствующий образом изменить содержимое аккумулятора, после чего вызвать подпрограмму в кодах.

(Очень подробно встроенные процедуры описаны в книге YAN LOGAN, FR. O'HARA "THE COMPLETE SPECTRUM ROM DISASSEMBLY", 1983, а также информацию о процедурах загрузки можно получить в ZX-РЕВЮ N5,6 1991г.)

Следует отметить, кроме всего вышеизложенного, что достаточно часто применяется комбинированная загрузка, когда наряду с блоками с хэдером загружаются блоки без хэдера.

Для того, чтобы читателю получить достаточно полное представление о структуре своей программы, рекомендуем загрузить ее в копировщик, который достаточно наглядно

покажет Вам из каких блоков состоит Ваша программа.

В копировщике ZX COPY 87, TF COPY и им аналогичным хэдер можно отличить достаточно легко потому, как именно в нем содержится информация о том, что это либо Бейсик, либо коды и т. д., а также само название файла. Разумеется, в копировщике Вы можете подробно изучить лишь стандартный хэдер, в то время как нестандартный подобному исследованию не поддается.

Структура хэдера.

Итак, теперь мы с Вами в общих чертах рассмотрели структуру фирменной программы. Конечно, каждая конкретная программа имеет свою структуру (в этом заключается ее оригинальность), но практически каждую из них можно классифицировать по способу загрузки - либо загружается файл с хэдером, либо загружающийся блок не имеет хэдера.

(Любопытно, что несколько лет назад появился оригинальный метод защиты программ от копирования. Он заключался в том, что в название программы наряду с обычными символами включались и ключевые слова Бейсика, либо управляющие коды. После загрузки этого "фальшхэдера" в копировщик, копировщик не мог распечатать эти символы и работа программы останавливалась с сообщением об ошибке. Великолепный комплект польских копировщиков ZX COPY 87, TF COPY, TF COPY 2, MR COPY и др. практически полностью свел на нет все усилия в данной области защиты. Интересным напоминанием этого направления защиты от копирования может служить программа GREEN BERET. Один из файлов которой длиной 17 байт как раз и должен был выполнять роль фальшхэдера.

Следует еще коротко сказать о том, как создатели копировальных программ вышли из положения. В последних разработках копировщиков вместо символов, не соответствующих ASCII кодам какой-либо литеры печатается вопросительный знак или иной заранее принятый символ.

Однако, несмотря на то, что метод, казалось бы, исчерпал себя, подробное его изучение может натолкнуть читателя на создание новых оригинальных разработок, базирующихся на этом или аналогичном принципах).

Теперь мы предлагаем Вам ознакомиться со структурой хэдера. Сделать это необходимо не только из чистого любопытства. Ведь невозможно создать что-то новое, не изучив досконально действующий старый образец.

Длина заголовка 19 байтов, а не 17, как написано в большинстве книг, но только 17 должны быть активны, поскольку "LOAD" (эта информация аналогична и для "SAVE") первый и последний байты определяют сами. Первый байт для заголовка всегда равен нулю, а последний - "байт четности" генерируется стандартной процедурой SAVE и потому нас не интересует.

Байт 2 содержит число, характеризующее тип записи, в зависимости от того, какое здесь содержится значение, компьютер определяет структуру файла, следующего за хэдером:

0 - это Бейсик-программа
1 - числовой массив
2 - массив символов
3 - блок кодов

Байты 3-12 содержат в себе имя программы

Байты 13 и 14 - длина основного блока (например для Бейсик программа это разность того, что содержится в системных переменных ELINE и PROG.

Байты 15 и 16 - начальный адрес загрузки кодов или номер строки автостарта для программ, написанных на Бейсике.

Байт 16 - для массива данных здесь располагается его имя в следующей форме:
Биты 0-4 - имя (от A=1 до Z=26)
Бит 5 - сброшен, если массив
Числовой бит 6 - активен, если массив

Строковый бит 7 - активен всегда

Байт 17 и 18 - длина Бейсик-программы, т. е. разность того, что содержится в системных переменных VARS и PROG.

Байт 19 - активизируется в ходе работы программ "LOAD" и "SAVE".

Для наглядности приведем схему заголовка (см. рис. 1):

Байты 1	2	3...12	13,14	15,16	17,18	19
Флаг	Тип	Имя	Длина	Старт	Длина для Бейсика	Четность
IX+...	0	1...10	11,12	13,14	15,16	17

Рис. 1 Структура "хэдера"

Поскольку хэдер анализируется интерпретатором с помощью индексной адресации, то внизу каждый байт дан как смещение относительно базового адреса, содержащегося в IX.

Небольшой комментарий по поводу некоторых байтов хэдера. В зависимости от считываемого блока байты 15 и 16 интерпретируются по-разному. В заголовках программ, написанных на Бейсике, эти байты содержат номер строки, с которой запускается программа, если она была записана с помощью оператора

```
SAVE "ИМЯ" LINE N
```

Если программа была записана без опции LINE и после считывания не запускается автоматически, то значение этого числа больше 32767. Любопытно, что одним из способов нейтрализации самозапускающихся программ на Бейсике является замена этих двух байтов на число большее 32767 (Подробно этот метод будет рассмотрен в одной из статей Главы 2 под названием "Блокировка автозапуска").

Байты 17 и 18 содержат число, указывающее длину самой программы на Бейсике, т. е. содержимое памяти от байта, указанного системной переменной PROG до байта, определяемого системной переменной VARS. Если мы от всей длины блока (байты 13 и 14) отнимем это число, то узнаем сколько байтов в этой программе занимают переменные Бейсика.

Теперь Вам достаточно хорошо известна структура хэдера и Вы уже представляете, чем является эта совокупность байтов. Однако, для того, чтобы исследовать хэдер более подробно, я рекомендую Вам использовать специальные программы, которые прозондируют загружающийся хэдер и выдадут информацию о нем.

Ниже в качестве примера приведена такая программа. Фактически она написана на Бейсике, но в своей работе обращается к процедурам в машинной коде. Программа загружает хэдер в определенное место памяти (которое, кстати, Вы можете изменить), после чего Бейсик-программа анализирует структуру хэдера и выдает всю информацию о ней на экран в виде письменных сообщений.

Сначала программа формирует процедуру в машинных кодах, используя блок DATA строки 30.

Фактически этот блок осуществляет запуск встроенной в ПЗУ подпрограммы-загрузчика, но делает это так, чтобы загружался только хэдер.

При загрузке хэдера в аккумуляторе процессора должен содержаться 0, а в регистре IX адрес, с которого начинает грузиться хэдер. Кроме этого, должен быть включен флаг переноса (флаг C регистра F).

```
1 REM TAPE EXAMINER
5 CLS:BEEP ,1,24:
      PRINT "LOAD A HEADER AND WAIT,PLEASE":
      PAUSE 150: BEEP ,1,24:CLS
7 RESTORE
10 CLEAR 32511
20 FOR A=32512 TO 32521:
        READ B:
```

```
        POKE A, B:
      NEXT A
30 DATA 175,55,221,33,16,127,205,86,5,201
40 LET B=32528:
      DEF FN A(X)=PEEK(B+X)*256*PEEK(B+X+1)
50 RANDOMIZE USR 32512
60 LET C=PEEK B
70 IF C>3 THEN GO TO 50
75 LET B$="":
      FOR A=B+1 TO B+10:
      LET B$=B$+CHR$ PEEK A:
      NEXT A
78 FOR A=10 TO I STEP -1
79 IF B$(A)=" "
      THEN LET B$=B$(TO A-1):
      NEXT A
80 IF B$=" " THEN LET B$=" "
85 PRINT "FILE NAME: ";INVERSE 1; B$
100 PRINT "FILE TYPE: "
110 GO SUB 1000+100*C
120 PRINT :PRINT
130 POKE B,255
140 GO TO 50
150 STOP
160 SAVE "TAPEXAN" LINE 5: CLS:
      PRINT "O. K. REWIND AND PLAY":
      VERIFY "TAPEXAN":
      CLS:
      PRINT "O.K. VERYFIED": STOP
1000 PRINT "PROGRAM"
1010 PRINT "TOTAL LENGTH: ":
      FN A(11); "BYTES"
1020 PRINT "PROGRAM LENGTH: ":
      FN A(15); "BYTES"
1030 IF FN A(13)>9999
      THEN PRINT "SAVED BY: "
      'TAB 5; "SAVE " " "; B$; " " " ":
      RETURN
1040 PRINT "SAVED BY:"
      'TAB 5; "SAVE " " ";
      B$; " " " LINE ";FN A(13)
1050 RETURN
1100 PRINT "NUMBER ARRAY"
1110 LET A$=" ":
      GOTO 1220
1200 PRINT "CHARACTER ARRAY"
1210 LET A$="$"
1220 PRINT "ARRAY LENGTH: ";FN A(11); "BYTES"
1230 LET D=PEEK (B+14)
1240 PRINT "ORIGINAL ARRAY NAME:";CHR$(64+32*(D/32-INT(D/32))) ;A$
1250 RETURN
1300 IF FN A(11)-6912 AND FN A(13)=16384 THEN PRINT "BYTES-SCREEN$": RETURN
1310 PRINT "BYTES"
1320 PRINT "START ADRESS: ";FN A(13)
1330 PRINT "LENGTH: ";FN A(11); "BYTES"
1340 RETURN
1350 CLEAR:
      SAVE "TAPEXAM" LINE 5:
      BEEP .1,24:
      PRINT AT 10,0: "O.K. REWIND AND PLAY TO VERIFY ":
      VERIFY "TAPEXAM":
      CLS:
      BEEP ,1,24:
      STOP
```

Листинг ассемблера программы в кодах, которая содержится в строке 30 DATA.

```
7F00   XOR A            Обнуление аккумулятора.
7F01   SCF              Принудительное включение флага переноса.
7F02   LD IX,7F10       В регистр IX поместили адрес с которого
                        начнется размещение хэдера в памяти компьютера.
7F06   CALL 0556        Вызов загружающей процедуры ПЗУ.
7F09   RET
```

Глава 2. Блокировка автозапуска.

Введение.

После внимательного ознакомления со структурой фирменных программ, читателю становится очевидно, что для взлома программы необходимо, прежде всего, исследовать Бейсик-файл (первичный загрузчик), этот вывод стал очевидным и для фирм-производителей программного обеспечения. Вот почему многие методы защиты направлены в основном на защиту Бейсик-файла.

Однако, следует заметить, что наряду с техникой программирования совершенствовалась и техника взлома, в этой области существует очень мало информации, но мой вывод подтверждает тот факт, что вскрытие хаккерами программ можно найти среди игр самых разнообразных годов выпуска.

В этой главе рассмотрены три концепции блокировки автозапуска программ. Естественно, каждую из этих концепций можно использовать независимо от других. Можно, однако применить и комплексный подход. Но основным является то, что каждый их предложенных Вашему вниманию подходов имеет свои достоинства, но он в то же время имеет и свои недостатки.

Мы думаем, что ознакомившись с этими концепциями более подробно. Вы выберете для себя ту, которая наилучшим образом удовлетворяет Вашим потребностям.

2.1 Загрузка Бейсика через блок кодов.

После того, как читатель ознакомился со структурой хэдера, ему становится ясно, что фактически между заголовком Бейсика и заголовком кодов разница небольшая. А разницы между непосредственным файлом Бейсика и непосредственным файлом кодов нет вообще никакой.

Примечание: В данном случае под файлом подразумевается та часть программы, которая загружается после хэдера. На самом деле отличия есть и заключаются они в следующем:

- Бейсик-файл загружается в память компьютера начиная с адреса, задаваемого системной переменной PROG, блок кодов же загружается с адреса, указанного в хэдере.
- Бейсик-файл обрабатывается интерпретатором как определенная последовательность символов-кодов Бейсика. Блок кодов обрабатывается процессором как последовательность кодов Z80.

На этом свойстве этих файлов и основан данный метод взлома. В самом деле, нам ничего не стоит обмануть компьютер - он думает, что загружает в память файл кодов, а на самом деле загружает файл Бейсика и при этом он не сможет установить никакой ошибки. Поскольку при той проверке, которую осуществляет "ZX SPECTRUM", такую ошибку установить просто-напросто невозможно.

Итак, рассмотрим этот метод более подробно.

Нам необходимо изучить структуру Бейсик-файла, при этом, чтобы избежать запуска команд и процедур, осуществляющих защиту, нам необходимо загрузить программу в память без автозапуска, т. е. так, чтобы она не стартовала, если в ней предусмотрена такая возможность.

Для того, чтобы решить данную задачу, нам необходимо прежде всего узнать длину Бейсик-файла. Это можно сделать, воспользовавшись вышеприведенной программой для анализа хэдера. То же можно сделать и воспользовавшись программой-копировщиком.

Обычно в копировщике мы можем наблюдать следующую картину (рис. 2):

Имя программы	Тип программы	Строка автозапуска	Длина
xxxxxxxxxxx	P	10	423
yyyyyyyyyyy	C	35000	22250
zzzzzzzzzzz	C	61200	1800

Рис. 2

После имени программы обычно появляется сообщение о типе данной программы: PROGRAM, CODE и т. п.

Строка автозапуска появляется только для программ Бейсика (для кодов же появляется номер ячейки памяти, с которого начинается загрузка). В разделе "длина" появляется длина загружаемого вслед за хэдером файла, это не длина хэдера.

Нам в нашей работе понадобится "реальная длина" (В большинстве случаев это значение не совпадает со значением "длина" в хэдере, но бывают и исключения). Кроме этого Вам необходимо запомнить номер строки автозапуска.

Следующим этапом нашей работы является создание нового хэдера, благодаря которому мы сможем загрузить нашу программу в произвольное место памяти. Как Вам уже вероятно известно, компьютер не в состоянии отличить файл Бейсика от файла кодов. Этим мы с Вами и воспользуемся. Мы создадим хэдер, который якобы должен загружать коды с определенного места памяти, но вместо файла кодов мы загрузим туда файл Бейсика для последующего изучения.

Для создания такого хэдера нам понадобится значение "реальной длины" файла, взятое из кодировщика. Теперь запишем на ленту этот хэдер кодов командой с клавиатуры: save "имя файла" code 30000, реальная длина

В графе "имя файла" Вам необходимо будет написать произвольное название файла по Вашему выбору, но не более 10 символов. Значение 30000 после CODE означает, что файл кодов, загружающийся после этого хэдера, будет загружаться с ячейки памяти 30000 (это же касается и записи информации т.е. после подачи данной команды, на ленту, должен будет записаться блок длиной "реальная длина", начиная с ячейки памяти 30000. Однако, поскольку нам необходим только хэдер, мы не будем дожидаться выгрузки на ленту файла кодов) и выключим магнитофон.

Теперь мы имеем следующие магнитофонные блоки:

Исходный хэдер (1) Бейсика	Файл Бейсика (1')	Специальный хэдер кодов (2)
Этот хэдер мы имели вместе с блоком Бейсик программы, длина хэдера 19 байт.	Этот файл описывался исходным хэдером и загрузить в компьютер автозапуском.	Этот хэдер мы создали таким образом, чтобы он описывал файл Бейсика как файл кодов. Длина=19 байт

Теперь нам необходимо непосредственно осуществить подмену. Для этого подадим команду с магнитофона:

```
LOAD "" CODE
```

после которой загрузим созданный нами хэдер. После этого хэдера Вам необходимо загрузить файл Бейсика (1'), однако этому файлу не должен предшествовать исходный хэдер (1). (В данном случае этот исходный хедер (1) не загружается вообще). Наилучшее техническое решение, позволяющее точно найти файл Бейсика (1'), пропустив исходный хэдер, является временное выдергивание шнура загрузки из магнитофона совместно с использованием клавиши ПАУЗА магнитофона (если такая имеется).

Итак, Вы осуществили подмену и загрузили Бейсик-файл под видом кодов. Теперь Вашей задачей является просмотреть этот файл и изучить его структуру.

(Продолжение следует.)

40 ЛУЧШИХ ПРОЦЕДУР

Продолжение.

Начало см. с. 17-28, 61-70.

7. ПРОЦЕДУРЫ ОБРАБОТКИ ПРОГРАММ.

7.1 Удаление блока программы.

Длина: 42
Количество переменных: 2
Контрольная сумма: 5977
Назначение: Эта программа удаляет блок BASIC-программы, находящийся между строками, определенными пользователем.
Переменные:
Имя - start line no
Длина - 2
Адрес - 23296
Комментарий: Номер первой строки, подлежащей удалению.
Имя - end line no
Длина - 2
Адрес - 23298
Комментарий: Номер последней строки, подлежащей удалению.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:
Если имеют место следующие ошибки, то процедура останавливается без удаления строк BASIC-программы:
- если последний номер строки меньше, чем первый номер строки;
- если между этими двумя строками нет программы на БЕЙСИКе;
- если один из номеров строк или оба равны 0.

Комментарий:
Эта программа довольно медленна для удаления большого блока программных строк, но, тем не менее, работать с ее помощью все же удобнее, чем удалять строки вручную.

Не вводите номера строк больше, чем 9999.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,(23296)      42     0      91
            LD DE,(23298)      237    91     2      91
            LD A,H             124
            OR L               181
            RET Z              200
            LD A,D             122
            OR E               179
            RET Z              200
            PUSH DE            213
            CALL 6510          205    110    25
            EX (SP),HL         227
```

```
          INC HL          35
          CALL 6510       205   110   25
          POP DE          209
          AND A           167
          SBC HL,DE       237   82
          RET Z           200
          RET C           216
          EX DE, HL       235
NXT_CH    LD A,D          122
          OR E            179
          RET Z           200
          PUSH DE         213
          PUSH HL         229
          CALL 4120       205   24    16
          POP HL          225
          POP DE          209
          DEC DE          27
          JR NXT_CH       24    243
```

Как она работает:

В пары регистров HL и DE загружаются начальный и конечный номера строк соответственно. Эти значения проверяются и, если одно из них или оба равны 0, программа возвращается в BASIC.

Затем вызывается подпрограмма ПЗУ по адресу 6510 - она возвращает адрес в памяти компьютера, с которого начинается первая строка. Эта же подпрограмма затем вызывается снова для определения адреса символа, стоящего после 'ENTER' в конечной строке.

В пару регистров HL помещается разность этих двух адресов и, если это значение равно 0 или отрицательно, программа возвращается в BASIC.

Содержимое пары регистров HL копируется в DE для использования DE в качестве счетчика. Если счетчик равен 0, то работа процедуры завершается, а если нет, то вызывается подпрограмма ПЗУ, расположенная по адресу 4120, которая удаляет один символ. После этого - возврат к 'NXT_CH'.

7.2 Обмен токена.

Примечание: под "токеном" подразумевается любое ключевое слово (команда, функция) из "словаря" БЕЙСИКа, которое рассматривается данной программой (как и интерпретатором компьютера) в виде определенного кода.

Длина: 46

Количество переменных: 2

Контрольная сумма: 5000

Назначение:

Меняет любое вхождение заданного токена в БЕЙСИК-программе на другой токен (например, все операторы PRINT могут быть изменены на LPRINT).

Переменные:

Имя - chr old

Длина - 1

Адрес - 23296

Комментарий: Код заменяемого токена.

Имя - chr new

Длина - 1

Адрес - 23297

Комментарий: Код замещающего токена.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если БЕЙСИК-программы нет в памяти или один из двух заданных токенов имеет код меньше 32, то процедура возвращается в БЕЙСИК.

Комментарий:

Эта процедура очень быстра, но чем длиннее БЕЙСИК-программа, тем медленнее она работает.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD BC,(23296)      237   75    0     91
            LD A,31            62    31
            CP B               184
            RET NC             208
            CP C               185
            RET NC             208
            LD HL,(23635)      42    83    92
NXT_CH      INC HL             35
            INC HL             35
            INC HL             35
CHECK       LD DE,(23627)      237   91    75    92
            AND A              167
            SBC HL,DE          237   82
            RET NC             208
            ADD HL,DE          25
            INC HL             35
            LD A,(HL)          126
            INC HL             35
            CP 13              254   13
            JR Z,NXT_CH        40    237
            CP 14              254   14
            JR NZ,COMP         32    3
            INC HL             35
            JR NEXT_CHR        24    230
COMP        DEC HL             43
            CP C               185
            JR NZ,CHECK        32    229
            LD (HL),B          112
            JR CHECK           24    226
```

Как она работает:

В регистры B и C загружаются новый и старый токены соответственно. Если любой из токенов имеет код меньше, чем 32, то программа возвращается в BASIC.

В пару HL заносится адрес начала БЕЙСИК программы. Пара регистров затем увеличивается и сравнивается с адресом области переменных. Если HL не меньше чем адрес начала области переменных программа возвращается в БЕЙСИК.

Пара HL увеличивается указывая на следующий символ. Код этого символа загружается в аккумулятор и HL увеличивается вновь. Если значение в аккумуляторе равно 13 или 14 (ENTER или NUMBER) подпрограмма возвращается к 'NXT_CH' а HL увеличивается указывая на следующий символ. Если аккумулятор не содержит 13 или 14, хранимое значение сравнивается с 'chr old'. Если пара найдена, этот символ замещается на 'chr new'.

Затем возврат к проверке ('CHECK') на конец обрабатываемой программы.

7.3 Удаление REM строк

Длина: 132

Количество переменных: 0

Контрольная сумма 13809

Назначение:

Эта программа удаляет все комментарии из БЕЙСИК программы.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если в памяти нет БЕЙСИК программы процедура вернется в БЕЙСИК без каких либо действий.

Комментарий:

Процедура ПЗУ, которая используется для удаления символов, выполняется не очень быстро и поэтому процедура "Удаление REM" работает медленно.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,(23635)      42    83    92
            JR CHECK           24    31
NEXT_L      PUSH HL            229
            INC HL             35
            INC HL             35
            LD C,(HL)          78
            INC HL             35
            LD B,(HL)          70
NXT_CH      INC HL             35
            LD A,(HL)          126
            CP 33              254   33
            JR C,NXT_CH        56    250
            CP 234             254   234
            JR NZ,SEARCH       32    26
            INC BC             3
            INC BC             3
            INC BC             3
            INC BC             3
            POP HL             225
DEL_L       PUSH BC            197
            CALL 4120          205   24    16
            POP BC             193
            DEC BC             11
            LD A,B             120
            OR C               177
            JR NZ,DEL_L        32    248
CHECK       LD DE,(23627)      237   91    75    92
            AND A              167
            SBC HL,DE          237   82
            RET NC             208
            ADD HL,DE          25
            JR NEXT_L          24    214
SEARCH      INC HL             35
            LD A,(HL)          126
            CP 13              254   13
            JR NZ,N_ENT        32    8
ENT_FD      POP HL             225
            ADD HL,BC          9
            INC HL             35
            INC HL             35
            INC HL             35
            INC HL             35
            JR CHECK           24    231
N_ENT       CP 14              254   14
            JR NZ,N_NUMB       32    7
            INC HL             35
            INC HL             35
            INC HL             35
```

```
N_NUMB      INC HL          35
            INC HL          35
            JR SEARCH       24    231
N_NUMB      CP 33           254   33
            JR C,SEARCH     56    227
            CP 34           254   34
            JR NZ,N_QUOT    32    8
FD_QD       INC HL          35
            LD A,(HL)       126
            CP 34           254   34
            JR NZ,FD_QT     32    250
            JR SEARCH       24    215
N_QUOT      CP 58           254   58
            JR NZ,SEARCH    32    211
            LD D,H          84
            LD E,L          93
FD_ENT      INC HL          35
            LD A,(HL)       126
            CP 13           254   13
            JR Z,ENT_FD     40    209
            CP 33           254   33
            JR C,FD_ENT     56    246
            CP 234          254   234
            JR NZ,N_QUOT    32    236
            LD H,D          98
            LD L,E          107
DEL_CH      PUSH BC         197
            CALL 4120       205   24    16
            POP BC          193
            DEC BC          11
            LD A,(HL)       126
            CP 13           254   13
            JR NZ,DEL_CH    32    245
            POP HL          225
            INC HL          35
            INC HL          35
            LD (HL),C       113
            INC HL          35
            LD (HL),B       112
            DEC HL          43
            DEC HL          43
            DEC HL          43
            JR CHECK        24    160
```

Как она работает:

В пару регистров HL загружается адрес начала БЕЙСИК области и делается переход к подпрограмме проверки конца обрабатываемой программы. Если конец достигнут, происходит возврат в БЕЙСИК.

Программа переходит к обработке следующей строки "NEXT_L". Адрес, имевшийся в HL сохраняется на стеке для дальнейшего использования, а в BC загружается длина обрабатываемой БЕЙСИК строки. Подпрограмма 'NXT_CH' увеличивает адрес в HL и загружает в аккумулятор очередной символ. Если его код меньше чем 33, т. е. это пробел или управляющий символ, процедура возвращается, чтобы повторить эту часть вновь. Если встретившийся символ не является символом REM, делается переход к 'SEARCH'.

Если оператор REM найден, регистр BC увеличивается на 4 (он может быть теперь использован в качестве счетчика), а HL восстанавливается из стека. Затем удаляется количество символов, определенное в BC, начиная с адреса определенного в HL. Для удаления используется процедура ПЗУ, расположенная по адресу 4120. Затем процедура снова переходит к процедуре 'CHECK'.

Если происходит переход к процедуре 'SEARCH', HL увеличивается указывая на следующий символ, и это значение загружается в аккумулятор. Если это символ ENTER, т.е.

достигнут конец строки, то HL восстанавливается из стека и увеличивается, указывая на начало следующей строки, а затем происходит переход к 'CHECK'.

Если аккумулятор хранит символ NUMBER (код равен 14), то HL увеличивается, указывая на первый символ после обнаруженного числа, и процесс поиска повторяется.

Затем делается проверка для символов, код которых меньше 33. Если такой символ обнаружен делается возврат к 'SEARCH'.

Если обнаружен символ "кавычка" (34), процедура выполняет цикл до тех пор, пока не будет обнаружена вторая кавычка, а затем поиск будет продолжен. Если найденный символ не двоеточие, определяющее строку с несколькими выражениями, поиск повторить.

HL копируется в DE, чтобы сохранить адрес двоеточия, а затем HL увеличивается, указывая на следующий символ. Если это символ ENTER, делается переход к 'ENT_FD', иначе, если это управляющий символ или пробел, программа возвращается к 'FD_ENT'.

Если символ не является кодом REM, происходит переход к 'N_QUOT'. Если же код REM найден, в HL загружается адрес последнего встретившегося двоеточия, а затем все символы от адреса, находящегося в HL до следующего символа ENTER удаляются. Указатели для этой строки корректируются, HL устанавливается в начало этой строки, и происходит переход к 'CHECK'.

7.4 Создание REM строк

Длина: 85

Количество переменных 3

Контрольная сумма 9526

Назначение:

Эта процедура создает выражение REM с заданным количеством символов в заданной строке. Символ выбирается пользователем.

Переменные:

Имя - line number

Длина - 2

Адрес - 23296

Комментарий: номер строки в которую надо вставить выражение REM.

Имя - number chr

Длина - 2

Адрес - 23298

Комментарий: Количество символов после REM.

Имя - chr code

Длина - 1

Адрес - 23300

Комментарий: Код символов после REM

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если номер строки равен 0, больше 9999, или строка с тем же самым номером уже существует, программа возвращается в BASIC.

Комментарий:

Эта программа не проверяет достаточно ли свободной памяти для добавления новой строки. Следовательно это должно быть сделано перед стартом с помощью программы "Размер свободной памяти" из нашей статьи. Символы для ввода после REM должны иметь коды больше чем 31, т. к. управляющие символы (0 - 31) могут сбить с толку подпрограмму LIST в ПЗУ.

Подпрограмма ПЗУ, которая вызывается для вставки символов, довольно медленно выполняется, занимая много времени.

Созданное выражение REM может быть использовано для хранения машинного кода или данных.

ЛИСТИНГ МАШИНЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ  ВВОДА

            LD HL,(23296)       42    0     91
            LD A,H              14
            OR L                181
            RET Z               200
            LD DE,10000         17    16    39
            AND A               167
            SBC HL,DE           237   82
            RET NC              208
            ADD HL,DE           25
            PUSH HL             229
            CALL 6510           205   110   25
            JR NZ,CREATE        32    2
            POP HL              225
            RET                 201
CREATE      LD BC,(23298)       237   75    2     91
            PUSH BC             197
            PUSH BC             197
            LD A,13             62    13
            CALL 3976           205   136   15
            INC HL              35
            POP BC              193
NXT_CH      PUSH BC             197
            LD A,B              120
            OR C                177
            JR Z,INSERT         40    11
            LD A,(23300)        58    4     91
            CALL 3976           205   136   16
            INC HL              35
            POP BC              193
            DEC BC              11
            JR NXT_CH           24    240
INSERT      POP BC              193
            LD A,234            62    234
            CALL 3976           205   136   15
            INC HL              35
            POP BC              193
            INC BC              3
            INC BC              3
            LD A,B              120
            PUSH BC             197
            CALL 3976           205   136   15
            POP BC              193
            INC HL              35
            LD A,C              121
            CALL 3976           205   136   15
            INC HL              35
            POP BC              193
            LD A,C              121
            PUSH BC             197
            CALL 3976           205   136   15
            POP BC              193
            INC HL              35
            LD A,B              120
            JP 3976             195   136   15
```

Как она работает:

В пару регистров HL загружается заданный номер строки. Это значение сравнивается с нулем и если он равен нулю, программа возвращается в БЕЙСИК. Если HL содержит число больше 9999 (максимально возможный номер строки), также происходит возврат в БЕЙСИК.

Вызывается подпрограмма ПЗУ (CALL 6510), которая возвращает в HL адрес строки, номер которой был в HL перед вызовом этой подпрограммы. Если флаг нуля установлен, то строка уже существует и в этом случае программа также возвращается в БЕЙСИК.

Если флаг нуля не установлен, делается переход к 'CREATE'. В BC загружается количество символов для вставки после 'REM' и это число запоминается на стеке. В аккумулятор загружается число 13 - код символа ENTER. Затем вызывается подпрограмма ПЗУ из адреса 3976 для вставки символа ENTER. Регистр BC восстанавливается из стека.

После повторного сохранения BC на стеке, пара BC тестируется чтобы определить есть ли еще символы для вставки. Если нет, то делается переход к 'INSERT'. Если есть еще символы для вставки, в аккумулятор загружается заданный нами код и подпрограмма по адресу 3976 (из ПЗУ) используется для его вставки. Счетчик BC уменьшается и программа возвращается к проверке BC на 0.

Как только процедура достигает 'INSERT', код REM вставляется, используя эту же подпрограмму. Затем в BC загружается длина созданной новой строки, и указатели длины для этой строки созданы. Номер строки затем извлекается из стека, и это значение вставляется перед возвратом в БЕЙСИК.

7.5 Сжатие программы.

Длина: 71

Количество переменных: 0

Контрольная сумма: 7158

Назначение:

Эта программа удаляет все ненужные управляющие символы и свободные пространства из БЕЙСИК-программы, увеличивая таким образом количество доступной памяти.

Вызов процедуры:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если в памяти нет БЕЙСИК-программы, процедура возвращается в БЕЙСИК.

Комментарий:

Эта процедура предполагает, что все выражения REM уже удалены из БЕЙСИК-программы. Если же это не так, произойдет сбой. Время выполнения пропорционально длине БЕЙСИК программы.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

           LD HL,(23635)      42    83    92
NEXT_L     INC HL             35
           INC HL             35
CHECK      LD DE,(23627)      237   91    75    92
           AND A              167
           SBC HL,DE          237   82
           RET NC             208
           ADD HL,DE          25
LENGTH     PUSH HL            229
           LD C,(HL)          78
           INC HL             35
           LD B,(HL)          70
NEXT_B     INC HL             35
LOAD       LD A,(HL)          126
           CP 13              254   13
           JR NZ,NUMBER       32    8
RESTOR     POP HL             225
           LD (HL),C          113
           INC HL             35
           LD (HL),B          112
```

```
        ADD HL,BC           9
        INC HL              35
        JR NEXT_L           24     227
NUMBER  CP 14               254    14
        JR NZ,QUOTE         32     7
        INC HL              35
        INC HL              35
        INC HL              35
        INC HL              35
        INC HL              35
        JR NEXT_B           24     231
QUOTE   CP 34               254    34
        JR NZ,CONTR         32     12
F_QUOT  INC HL              35
        LD A,(HL)           126
        CP 34               254    34
        JR Z,NEXT_B         40     221
        CP 13               254    13
        JR Z,RESTOR         40     223
        JR F_QUOT           24     244
CONTR   CP 33               254    33
        JR NC,NEXT_B        48     211
        PUSH BC             197
        CALL 4120           205    24     16
        POP BC              193
        DEC BC              11
        JR LOAD             24     204
```

Как она работает:

В пару регистров HL загружается адрес БЕЙСИК-программы. HL затем увеличивается дважды, указывая на два байта, хранящих длину следующей строки. В пару регистров DE загружается адрес области переменных. Если HL не меньше, чем DE, подпрограмма возвращается в BASIC, т. к. достигнут конец программной области.

Адрес из HL сохраняется в стеке, в BC загружается длина строки, и HL увеличивается, указывая на следующий байт в строке. Байт в HL затем загружается в аккумулятор. Если это значение не равно 13, происходит переход к 'NUMBER'.

Для достижения 'RESTOR' должен быть найден конец строки. Адрес указателей длины строки загружается из стека в HL и вставляется длина существующей строки. Длина строки добавляется к HL, HL увеличивается и программа возвращается к NEXT_L.

Если программа достигает 'NUMBER', проверяется, содержит ли аккумулятор символ NUMBER (код 14). Если да, HL увеличивается на 5 - т. е. выполняется прыжок через следующее за NUMBER число, и происходит переход к 'NEXT BYTE'.

Если аккумулятор не содержит кода кавычки, программа переходит к 'CONTR'. Если код кавычки найден, циклы программы будут выполняться до тех пор, пока не будет достигнут конец строки или найдена другая кавычка. В первом случае происходит переход к RESTOR, в последнем - к NEXT_B.

В процедуре CONTR происходит проверка символа. Если он имеет код не меньше, чем 33, подпрограмма возвращается к 'NEXT_B'.

Если свободное пространство или управляющий символ найдены, вызывается подпрограмма ПЗУ по адресу 4120 для его удаления. Длина строки, которая хранится в BC, уменьшается и происходит переход к 'LOAD'.

7.6 Загрузка машинного кода в DATA-строку.

Длина: 179

Количество переменных: 2

Контрольная сумма: 19181

Назначение:

Эта программа создает выражение DATA в первой строке программы на БЕЙСИКе, а

затем заполняет его данными из памяти.

Переменные:

Имя - data start

Длина - 2

Адрес - 23296

Комментарий: Адрес данных для копирования.

Имя - data length

Длина - 2

Адрес - 23298

Комментарий: Количество байтов для копирования.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если число байтов для копирования равно 0, или первая строка уже есть, программа возвращается в БЕЙСИК. Программа не проверяет, достаточно ли памяти для новой строки. Программа требует 10 байтов на байт данных плюс пять байтов для номеров строк, указателей длины и т. д. Кроме того, используемая подпрограмма ПЗУ также использует большую рабочую область - принимайте это в расчет. Если доступной памяти недостаточно, указатели строки будут установлены неправильно, и листинг БЕЙСИКа будет недостоверен.

Комментарий:

Время, занимаемое работой этой программы, пропорционально объему памяти для копирования.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD DE,(23296)      237   91    0     91
            LD BC,(23298)      237   75    2     91
            LD A,B             120
            OR C               177
            RET Z              200
            LD HL,(23635)      42    83    92
            LD A, (HL)         126
            CP 0               254   0
            JR NZ,CONT         32    6
            INC HL             35
            LD A,(HL)          126
            CP 1               254   1
            RET Z              200
            DEC HL             43
CONT        PUSH HL            229
            PUSH BC            197
            PUSH DE            213
            SUB A              151
            CALL 3976          205   136   15
            EX DE,HL           235
            LD A,1             62    1
            CALL 3976          205   136   15
            EX DE,HL           235
            CALL 3976          205   135   15
            EX DE,HL           235
            CALL 3976          205   135   15
            EX DE,HL           235
            LD A,228           62    228
            CALL 3976          205   135   15
            EX DE,HL           235
NEXT_B      POP DE             209
            LD A,(DE)          26
            PUSH DE            213
```

```
         LD C,47          14    47
HUNDR    INC C            12
         LD B,100         6     100
         SUB B            144
         JR NC,HUNDR      48    250
         ADD A,B          128
         LD B,A           71
         LD A, C          121
         PUSH BC          197
         CALL 3976        205   136   15
         EX DE,HL         235
         POP BC           193
         LD A,B           120
         LD C,47          14    47
TENS     INC C            12
         LD B,10          6     10
         SUB B            144
         JR NC,TENS       48    250
         ADD A,B          128
         LD B, A          71
         LD A,C           121
         PUSH BC          197
         CALL 3976        205   136   15
         POP BC           193
         EX DE,HL         235
         LD A,B           120
         ADD A,48         198   48
         CALL 3976        205   136   15
         EX DE,HL         235
         LD A,14          62    14
         LD B, 6          6     6
N_ZERO   PUSH BC          197
         CALL 3976        206   136   15
         POP BC           193
         EX DE,HL         235
         SUB A            151
         DJNZ N_ZERO      16    247
         POP DE           209
         PUSH HL          229
         DEC HL           43
         DEC HL           43
         DEC HL           43
         LD A,(DE)        26
         LD (HL),A        119
         POP HL           225
         INC DE           19
         POP BC           193
         DEC BC           11
         LD A,B           120
         OR C             177
         JR Z,ENTER       40    10
         PUSH BC          197
         PUSH DE          213
         LD A,44          62    44
         CALL 3976        205   136   15
         EX DE,HL         235
         JR NEXT_B        24    173
         LD A,13          62    13
ENTER    CALL 397         205   136   15
         POP HL           225
         LD BC,0          1     0     0
         INC HL           35
         INC HL           35
         LD D,H           84
         LD E,L           93
```

```
         INC HL            35
POINT    INC HL            35
         INC BC            3
         LD A,(HL)         126
         CP 14             254   14
         JR NZ,END?        32    12
         INC BC            3
         INC BC            3
         INC BC            3
         INC BC            3
         INC BC            3
         INC HL            35
         INC HL            35
         INC HL            35
         INC HL            35
         INC HL            35
         JR POINT          24    237
END?     CP 13             254   13
         JR NZ,POINT       32    233
         LD A,C            121
         LD (DE),A         18
         INC DE            19
         LD A,B            120
         LD (DE),A         18
         RET               201
```

Как она работает:

В пару регистров DE загружается адрес байтов для копирования, а в пару регистров BC загружается число байтов для копирования. Если BC содержит 0, программа возвращается в БЕЙСИК.

В пару регистров HL загружается адрес БЕЙСИК-программы. В аккумулятор загружается байт из адреса установленного в HL. Это старший байт номера строки. Если он не равен 0, первая строка не существует и программа переходит к 'CONT'. Если старший байт содержит 0, в аккумулятор загружается младший байт. Если это значение установлено в 1, первая строка уже существует и программа возвращается в БЕЙСИК.

Адрес старшего байта номера строки сохраняется на стеке. Сохраняется число байтов для копирования, затем адрес данных. В аккумулятор загружается 0 (старший байт нового номера строки). Вызванная подпрограмма ПЗУ, находящаяся по адресу 3976, вставляет символ, имеющийся в аккумуляторе, по адресу установленному в HL.

В HL восстанавливается значение, хранившееся там перед этой операцией. В аккумулятор загружается 1 и это значение вставляется 3 раза. Первая единица - это младший байт номера строки, следующие две - указатель длины строки. В аккумулятор затем загружается код символа 'DATA' и это значение вводится в строку.

Адрес следующего байта данных восстанавливается из стека и загружается в DE. Сам этот байт загружается в аккумулятор, а содержимое DE вновь сохраняется на стеке. В C-регистр загружается значение, на 1 меньшее, чем код символа "0". C-регистр увеличивается, а в B-регистр загружается значение 100. B-регистр вычитается из аккумулятора и, если результат не отрицательный, подпрограмма возвращается к 'HUNDR'.

B-регистр прибавляется к аккумулятору. Таким образом аккумулятор содержит положительное значение. Это значение затем загружается в B-регистр. В аккумулятор загружается содержимое C-регистра, а BC сохраняется на стеке. Вызовом подпрограммы ПЗУ (3976), вставляется символ, хранящийся в аккумуляторе, по адресу, содержащемуся в HL. Пара регистров BC восстанавливается из стека, а в аккумулятор загружается значение B-регистра. Вышеупомянутый прием затем повторяется для B=10. Аккумулятор затем увеличивается на 48 и полученный в результате символ вставляется в строку.

Вышеописанная подпрограмма делала вставку десятичного представления встречающегося байта данных в выражение DATA. Теперь должно быть вставлено двоичное представление. Это значение маркируется с помощью кода NUMBER (CHR 14), который

вводится первым, а за ним 5 нулей. Значение копируемого байта помещается в ячейку, чтобы заместить третий ноль. DE увеличивается, указывая на следующий байт данных. Число байтов для копирования снимается из стека в BC, и это значение уменьшается. Если результат равен 0, делается переход к 'ENTER', иначе содержимое пар регистров BC и DE вторично помещается в стек, в выражение DATA включается запятая, а программа возвращается к 'NEXT_B'. В данной процедуре код ENTER (конец строки) добавляется для маркировки конца выражения DATA. В HL загружается адрес начала строки, а BC устанавливается в 0. HL увеличивается, указывая на младший байт указателя строки, и этот новый адрес копируется в DE. HL увеличивается, указывая на старший байт указателя строки. HL и BC затем увеличиваются, а в аккумулятор загружается символ, находящийся по адресу в HL.

Если аккумулятор содержит код 14, число найдено, то HL и BC увеличиваются на 5, перепрыгивая через число и указывая на символ, следующий за ним. Подпрограмма затем переходит к 'POINT'.

Если аккумулятор не содержит значений 14 и 13, происходит переход к 'POINT'.

Завершение формирования строки выполняется помещением содержимого BC в ячейку указателя длины строки. На необходимый адрес указывает DE. По окончании работы процедура выполняет возврат в БЕЙСИК.

8. ИНСТРУМЕНТАЛЬНЫЕ ПРОГРАММЫ

8.1 Определение размера свободной памяти.

Длина: 14
Количество переменных: 0
Контрольная сумма: 1443
Назначение:
Дает количество свободного пространства ОЗУ в байтах.
Вызов программы.

```
PRINT USR адрес
```

Контроль ошибок: Нет
Комментарий:

Эта программа должна вызываться перед использованием любых подпрограмм, которые могут увеличивать длину программы, чтобы быть уверенным в том, что в ОЗУ достаточно свободного пространства.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

           LD HL,0             33    0     0
           ADD HL,SP           57
           LD DE,(23653)       237   91    101   92
           AND A               16
           SBC HL,DE           237   82
           LD B,H              68
           LD C,L              77
           RET                 201
```

Как она работает:

В пару регистров HL загружается 0 и это значение суммируется с адресом конца свободной области ОЗУ (адрес хранится в SP). Пара регистров DE загружается адресом начала свободной области ОЗУ и вычитается из HL. HL копируется в BC и программа возвращается в БЕЙСИК.

8.2 Определение длины БЕЙСИК-программы

Длина: 13
Количество переменных: 0
Контрольная сумма: 1544
Назначение:
Дает длину БЕЙСИК-программы в байтах.
Вызов программы:
```
PRINT USR адрес
```
Контроль ошибок: Нет
Комментарий: Нет

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

           LD HL,(23637)      42    75    92
           LD DE,(23635)      237   91    83    92
           AND A              167
           SBC HL,DE          237   82
           LD B,H             68
           LD C,L             77
           RET                201
```

Как она работает:
В пару регистров HL загружается адрес области переменных, а в DE загружается адрес БЕЙСИК-программы. DE вычитается из HL, чтобы получить длину программы. HL копируется в BC и программа возвращается в БЕЙСИК.

8.3 Определение адреса БЕЙСИК-строки.

Длина: 29
Количество переменных: 1
Контрольная сумма: 2351
Назначение:
Возвращает адрес первого символа после кода 'REM' в заданной строке.
Переменные:
Имя - line number
Длина - 2
Адрес - 23296
Комментарии: номер строки, которая должна содержать 'REM'.
Вызов программы:
```
LET A = USR адрес
```
Контроль ошибок:
Если заданная строка не существует, или нет выражения REM, программа возвратит значение 0.
Комментарии:
Эта программа может быть использована для нахождения адреса ячейки, в которую может быть помещен машинный код. После выполнения программы переменная A (может быть использована любая переменная) устанавливается по адресу, или в 0, если имеет место ошибка.
Не вводить номер строки более 9999!

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

           LD HL,(23296)      42    0     91
```

```
          LD A,H           124
          OR L             181
          JR Z,ERROR       40    5
          CALL 6510        205   110   25
          JR Z,CONT        40    4
ERROR     LD BC,0          1     0     0
          RET              201
CONT      INC HL           35
          INC HL           35
          INC HL           35
          INC HL           35
          LD A,234         62    234
          CP (HL)          190
          JR NZ,ERROR      32    243
          INC HL           35
          LD B,H           68
          LD C,L           77
          RET              201
```

Как она работает:

В пару регистров HL заносится определенный номер строки. Если этот номер равен 0, делается переход к 'ERROR', иначе вызывается подпрограмма ПЗУ по адресу 6510. По возвращении из этой процедуры HL содержит адрес строки. Если флаг нуля установлен, делается переход к 'CONT'. Если флаг нуля не установлен, эта строка не существует, и подпрограмма переходит к 'ERROR', где в BC загружается 0, и затем происходит возврат в BASIC.

Если программа достигает метки 'CONT', HL увеличивается на 4, чтобы указать на первый оператор в данной строке. Если этот оператор не имеет кода 234, происходит переход к 'ERROR'. Если же это оператор 'REM', HL увеличивается, указывая на следующий символ. Значение HL затем копируется в BC и программа возвращается в БЕЙСИК.

8.5 Копирование данных в памяти.

Длина: 33

Количество переменных: 3

Контрольная сумма: 4022

Назначение:

Эта программа копирует область памяти с одного места в другое.

Переменные:

Имя - start

Длина - 2

Адрес - 23296

Комментарий: адрес источника для копирования.

Имя - destination

Длина - 2

Адрес - 23298

Комментарий: Адрес, в который происходит копирование.

Имя - length

Длина - 2

Адрес - 23300

Комментарий: длина блока, подлежащего копированию.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий:

Эта подпрограмма может быть использована для создания "мультипликации" с помощью следующего метода:

- создание первого экрана информации;
- копирование экрана выше RAMTOP;

- повторить для других экранов;
- копирование экранов в обратном направлении по одному в быстрой последовательности.

ЛИСТИНГ МАШИННЫХ КОДОВ

МЕТКА	АССЕМБЛЕР	ЧИСЛА ДЛЯ ВВОДА			
	LD HL,(23296)	42	0	91	
	LD DE,(23298)	237	91	2	91
	LD BC,(23300)	237	75	4	91
	LD A,B	120			
	OR C	177			
	RET Z	200			
	AND A	157			
	SBC HL,DE	237	82		
	RET Z	200			
	ADD HL,DE	25			
	JR C,COPY	56	3		
	LDIR	237	176		
	RET	201			
COPY	EX DE,HL	235			
	ADD HL,BC	9			
	EX DE,HL	235			
	ADD HL,BC	9			
	DEC HL	43			
	DEC DE	27			
	LDDR	237	184		
	RET	201			

Как она работает:

В пару регистров HL загружается адрес первого байта памяти для копирования, в DE загружается адрес, в который копируется память, а в BC загружается количество байтов для копирования. Если BC=0 или HL=DE, то подпрограмма возвращается в БЕЙСИК. Если HL больше, чем DE, часть памяти копируется, используя инструкцию 'LDIR', и программа возвращает в БЕЙСИК.

Если же DE больше, чем HL, то к обеим парам регистров прибавляется по BC-1 и память копируется, используя инструкцию 'LDDR', после чего программа возвращается в БЕЙСИК.

Окончание следует.

ВОЗВРАЩАЯСЬ К НАПЕЧАТАННОМУ

КАНАЛЫ И ПОТОКИ

В 12-м номере "ZX-РЕВЮ" мы подняли вопрос о концепции каналов и потоков. Среди наших читателей есть мнение о том, что при всей нужности и важности этого вопроса, сама статья грешила определенной академичностью и, если читать ее от начала и до конца, то не все могли с ней разобраться. Большинство взяло то, что лежало на поверхности, а остальное отложили для разбора на долгое время.

В связи с этим мы с радостью принимаем предложение, высказанное нашим постоянным корреспондентом из г. Балашова Саратовской области Пашориным В. Г. о введении постоянной рубрики "Возвращаясь к напечатанному", и начинаем с того, что дадим рекомендации, которые предлагает наш уважаемый автор для тех, кто хотел бы не только в теории ознакомиться поближе с каналами и потоками, но и применить свои знания на практике.

Это достойное развитие начатой темы, и нам будет очень приятно, если полезное начинание будет подхвачено и новая рубрика вызовет появление новых публикаций.

* * *

Для начала небольшое уточнение. Информация о существующих каналах хранится в ПЗУ и только после включения компьютера и инициализации системы эта информация копируется в область, о которой указывалось в статье - непосредственно перед областью Бейсик-программы. А этот участок памяти, как известно, является уже частью ОЗУ. Такое дублирование области информации о каналах из ПЗУ в ОЗУ и одновременное существование двух одинаковых областей не является непродуманной расточительностью памяти. Это глубоко продуманное решение, ориентированное на активных пользователей и разработчиков программного обеспечения, после инициализации система работает только с областью, находящейся в ОЗУ. А как известно, из ОЗУ можно не только читать информацию, но и записывать туда свою информацию. Это значит, что внося продуманные изменения в существующую область информации о каналах или расширяя ее для создания новых, пользовательских, каналов, можно получить интересные эффекты. Для примера выберем канал S. Информация об этом канале находится в ячейках с 23739 по 23743 (см. Табл. 1)

Таблица 1.

Адрес	Содержимое	
23739	244	9*256+244=2548 - адрес процедуры
23740	9	обслуживания канала при выводе
23741	196	21*256+196=5572 - адрес процедуры
23742	21	обслуживания канала при вводе
23743	83	код литеры S

Во многих фирменных программах для того, чтобы не искажать изображение на экране, полученное после дисплейного файла, перед загрузкой последующих блоков программы меняют адрес процедуры обслуживания канала S при выводе путем записи простых команд:

```
POKE 23739,82
POKE 23740,0
```

При этом загрузка блока будет выполняться, но на экране не будет сопровождающих надписей:

```
Program...
Byte...
```

и т.д.

Использование же команды CL0SE#2 для этих целей неприемлемо. После загрузки

всех блоков программы адрес процедуры вывода восстанавливается:

```
POKE 23739,244
POKE 23740,9
```

Этот нехитрый прием могут применять пользователи и для своих целей.

Другое, не менее интересное применение концепции каналов и потоков для практических целей - это изменение функции канала R. Основное его назначение это ввод данных из буфера редактора. Канал нельзя обслуживать, программируя на языке Бейсик, но его можно изменить так, чтобы редакция строк была невозможна. Это будет интересно тем, кто работает над защитой своих программ от несанкционированного вмешательства, достаточно записать:

```
POKE 23744,124
POKE 23745,0
```

и редакция строки будет невозможна.

Аналогичные эффекты можно получить, манипулируя не с данными в области информации о каналах, а с данными в системной переменной STRMS (23568). Так как каналы связаны с определенными потоками, то переподключая потоки к другим каналам, получим новые эффекты. Ну, например, чтобы запретить вывод на экран информации о программе при загрузке ее с магнитофона, достаточно записать в ячейку 23570 через команду POKE вместо числа 6 число 16, переподключив поток-2, связанный с каналом S к каналу P, обеспечивающему вывод на принтер. Переподключать стандартные каналы к потокам можно и с помощью hash-символа #. Попробуйте ввести программу (табл. 2)

Таблица 2

```
10 PRINT # 0; "Текст в нижней части экрана": PAUSE 0
20 LPRINT # 2; "Текст в верхней части экрана": PAUSE 0
30 LPRINT # 3; "Текст на принтере": PAUSE 0
40 LIST # 1
```

и Вы убедитесь, что знакомые Вам Бейсик-инструкции работают несколько необычно.

Кроме системных переменных CHANS(23631/2) и STRMS(23568), о которых упоминалось в статье, существует еще одна системная переменная (а о ней ничего не упоминалось), сохраняющая информацию о каналах. Эта переменная CURCHL (23633/4) Current Channel, в которой записан адрес активизированного в данный момент канала. Именно значение этой переменной используется инструкцией RST 16 для вывода информации. Кроме того, системные переменные FLAGS (23611), FLAGS 2(23658) и TV FLAG (23612) также имеют некоторое отношение к каналам и потокам.

Внимательный пользователь заметил, что для каналов K, S и P процедура вывода имеет один и тот же адрес - 2548. Так вот, бит 1 переменной FLAGS указывает на то, должен ли символ выводится на принтер (бит равен 1) или на экран (бит равен 0).

Нулевой бит переменной TV FLAG информирует о том, используется ли верхняя часть экрана (бит равен 0) или нижняя (бит равен 1). Ну и бит 4 переменной FLAGS 2, находясь в различных состояниях, (сброшен - установлен), определяет является ли канал к рабочим (бит установлен) или нерабочий в данный момент (бит сброшен).

Прежде, чем перейти к процедурам, демонстрирующим возможность использования концепции каналов и потоков на компьютерах с 48K ОЗУ, еще одно замечание. Поскольку, как отмечалось выше, информация о каналах хранится и в ПЗУ и в ОЗУ, то можно, например, увеличить область Бейсик-программы, исключив область информации о каналах из ОЗУ. Для этого надо переписать значение системных переменных:

```
10 POKE 23635, PEEK 23631
20 POKE 23636, PEEK 23632
30 POKE 23631, 175
40 POKE 23632, 21
```

Здесь в строках 10 и 20 меняется значение системной переменной, указывающей на начало Бейсик-программы. Теперь она будет начинаться не с адреса 23755, а с адреса 23734. В строках же 30 и 40 в системную переменную CHANS записывается начальный адрес области информации о каналах, находящейся в ПЗУ. Таким образом, участок памяти

для хранения Бейсик-программы увеличивается на 21 байт.

Весьма полезной, особенно для создателей обучающих программ, может быть следующая процедура, которая несколько меняет действие команды PRINT. При использовании этой процедуры по команде PRINT информация на экран будет выводиться с временной паузой между отдельными символами и с генерацией короткого звука после вывода каждого символа. Причем величина временной паузы между символами может быть эффективно использована при создании обучающих программ, например, по чтению или скорочтению. Эффект имитации работы пишущей машинки или телетайпа, кроме того, вносит разнообразие в работу программы (вспомните, например, игровую программу Black Hawk).

А вот и сама процедура:

```
 10   BEEP      EQU   949
 20   CURCHL    EQU   23633
 30   TIME      EQU   65300
 40             ORG   55301
 50   PRINT     DEFW  2548
 60   START     PUSH  AF
 70             LD    A,I
 80             JR    PO,NOPAUSE
 90             LD    A,(TIME)
100             LD    B,A
110   PAUSE     HALT
120             DJNZ  PAUSE
130   NOPAUSE   POP   AF
140             CP    33
150             JR    C,NOBEEP
160             PUSH  AF
170             PUSH  IX
180             LD    DE,50
190             LD    HL,100
200             CALL  BEEP
210             POP   IX
220             POP   AF
230   NOBEEP    LD    HL,(PRINT)
240             CALL  111
250             LD    HL,(CURCHL)
260             LD    BC,START
270             LD    E,(HL)
280             LD    (HL),C
290             INC   HL
300             LD    D,(HL)
310             LD    (HL),B
320             LD    A,B
330             CP    D
340             JR    NZ,CHNG
350             LD    A,C
360             CP    E
370             RET   Z
380             LD    (PRINT),DE
390             RET
```

Запуск этой процедуры может показаться несколько необычным, так как для этого не используется функция вызова программы в машинных кодах USR. Для того, чтобы эта процедура выполнялась, достаточно записать в область информации о каналах (в ячейки, отведенные для адреса процедуры вывода, обслуживающей канал S), адрес созданной процедуры. Стандартно тем записан адрес 2548, а мы с помощью команд POKE записываем туда стартовый адрес созданной процедуры - 65303:

```
POKE 23739, 23
POKE 23740,255
```

Теперь по команде RST 16 (вызов процедуры вывода символа на экран) будет вызываться не процедура из ПЗУ, а из адреса 65303. Временная пауза между выводимыми символами задается путем записи с помощью команды POKE соответствующего числа (от 1

до 10) в ячейку 65300.

Пояснения к процедуре: в строках 50...80 выполняется проверка на разрешение прерывания. Если прерывания запрещены, то выполняется переход к строке с меткой NOPAUSE и обход инструкции HALT, приводящей к зависанию компьютера в этом случае.

- В строках 90...120 формируется требуемая временная пауза перед выводом на экран очередного символа.

- в строках 130...150 выполняется проверка кода выводимого символа.

Если код менее 33, то это не печатаемый символ, а управляющий код и выполняется переход к строке с меткой NOBEEP, минуя строки 160...220, обеспечивающие выдачу звукового сигнала вместе с выводом символа на экран. Поскольку обработка управляющих кодов происходит с использованием других процедур, то прямой вызов CALL 2548 может привести к ошибке или потере управляющего кода. Поэтому выполняется вызов CALL 111. По адресу 111 записана команда JP (HL). Таким образом мы реализуем несуществующую в ассемблере процессора Z-80 команду CALL (HL) и выполняем переход по адресу, указанному в регистровой паре HL (строка 130).

Внутри процедур обработки управляющих кодов имеются команды, которые меняют содержимое байтов области информации о каналах. Поэтому в строках 250...310 выполняется проверка записанного для канала S адреса процедуры вывода и восстановление адреса 65303, а затем восстановление, при необходимости, переменной PRINT (строки 320...390).

Предложенная Вашему вниманию процедура прекрасно работает только до момента, пока не будет выполнена команда CLS, так как эта команда восстанавливает в области информации о каналах стандартные адреса процедур вывода. Чтобы вернуть прежнее действие команде PRINT, необходимо опять же с помощью команд POKE записать в ячейки 23739/40 адрес 65303. И так поступать после каждой команды CLS или нажатия клавиши ENTER.

Естественно, это не совсем удобно, поэтому лучшим решением будет открытие нового пользовательского канала, адреса процедур ввода-вывода которого будут оставаться незыблемыми, пока включен компьютер. Для этого необходимо расширить область информации о каналах на 5 байтов, в которых записать адреса соответствующих процедур ввода и вывода и имя нового канала. Выполняется это с помощью процедуры ПЗУ, находящейся по адресу 5717. При этом перед вызовом этой процедуры в регистровую пару HL необходимо записать начальный адрес резервируемой области, а в регистровую пару BC - количество резервируемых байтов. При использовании этой процедуры автоматически переписывается содержимое системных переменных, определяющих адреса отдельных блоков Бейсик-области. Ниже представлена программа, позволявшая создать новый канал:

```
10 FOR N=23296 TO 23304: READ A: POKE N,A: NEXT N
20 DATA 1,5,0,33,202,92,195,85,22
30 RANDOMIZE USR 23296
40 RESTORE 60
50 FOR N=23754 TO 23758: READ B: POKE N, B: NEXT N
60 DATA 23,255,196,21,83
70 STOP
```

После выполнения этой программы останется только подключить канал к потоку 2 командой POKE 23578,21. Теперь можно не беспокоиться о необходимости восстанавливать адрес созданной процедуры в области данных о каналах.

Алексеев А. Г.

ПРОФЕССИОНАЛЬНЫЙ ПОДХОД

Уважаемые читатели, если Вы читаете "ZX-РЕВЮ" не первый год, то должны быть знакомы с серией статей Стива Тернера под названием "Профессиональный подход", печатавшихся в прошлом году.

При всем уважении к английской школе программирования для "Спектрума" мы должны сказать что и у нас немало хороших программистов. Только скромность не позволяет честно признать что для "Спектрума" у нас их сейчас может быть уже и больше, а не видно их работы только потому что нет инфраструктуры многочисленных фирм способных приобретать их разработки и тиражировать массовыми тиражами.

Нет пока у нас и средств массовой информации, способных объединить миллионы людей, охваченных единой страстью Мы прекрасно понимаем что роль ZX-РЕВЮ в этом деле конечно какая-то есть, но это не более, чем капля в море. Мы ведь не охватываем и сотой доли процента тех, кто в этом нуждается.

В нашем портфеле с каждым днем становится все больше и больше интересных статей от отечественных программистов, которым есть чем поделиться с многочисленными любителями, как с начинающими, так и с имеющими некоторый опыт.

Сегодня мы возвращаемся к рубрике "Профессиональный подход", но уже на другом уровне. Теперь мы открываем ее для наших отечественных специалистов, которым есть что сказать своим коллегам. Если и у Вас есть идеи и Вы можете их изложить доступно для понимания широкими массами, мы открыты и для вас. Работа оплачивается.

ОБРАБОТКА ОШИБОК В БЕЙСИКЕ

Среди программистов более или менее освоивших Бейсик, прошитый в ПЗУ "Спектрума", бытует мнение, что невозможно на Бейсике создать хорошую "качественную" программу, которая к тому же выглядела бы достаточно "фирменно". Хорошая программа - на 5 баллов – это, как правило, программа в машинных кодах. Существует также достаточно много расширений возможностей обычного Бейсика. Это LASER-BASIC, BETA-BASIC, MEGA-BASIC и другие. Однако все их надо догружать с ленты, при этом результирующая программа получается достаточно громоздкой. Эти расширения Бейсика хороши и оправданы в том случае, если Вы создаете например серьезную игровую программу, заведомо претендующую на "фирменность". Зачастую же для повседневных задач не нужны богатые возможности расширений Бейсика. Прелесть Бейсика-ПЗУ в том, что он всегда сразу же готов к работе.

Что такое "повседневные задачи"? Ну, например я применяю компьютер для обучения детей устному счету, обучения азбуке. Одно дело, когда этому учит мама или папа, и совсем другое дело для ребенка вести диалог с КОМПЬЮТЕРОМ! Эффект обучения здесь на порядок выше. Компьютер может применяться в школе с первого до последнего класса. (На эту тему можно говорить бесконечно.) А студентам очень пригодится, например, при расчетах курсовых проектов. Это и есть "повседневные задачи". Вы освоили Бейсик ПЗУ и в состоянии написать нехитрую программу для той задачи, которая сейчас нужна. Задача решена и программа, может быть, больше никогда не понадобится. А может быть Ваш труд и не пропадет даром, а заинтересует других. Но это произойдет вероятнее в том случае, если программа выглядит "фирменно". Помните: встречают по одежке. Я применяю несколько простых приемов, которыми готов поделиться. Освоив их, Вы сможете придать "фирменный" вид любой самой простой и примитивной Бейсик-программе. Да Вы и сами получите удовольствие от пользования такой программой.

Итак, первый "фирменный" прием.

ПРЕДОТВРАЩЕНИЕ ОСТАНОВКИ БЕЙСИК ПРОГРАММЫ

Мало кому нравятся Бейсик программы, которые останавливаются при первой встретившейся ошибке или случайном нажатии клавиши "BREAK" Кроме неудобства в работе, такие программы выглядят "нефирменно", в отличие от программ в машинных кодах. Однако существует способ, при помощи которого можно устранить остановку программы от клавиши "BREAK" или при встретившейся ошибке, причем в зависимости от ошибки и места в программе, где эта ошибка произошла, возможна различная, заранее запланированная реакция программы.

Этот способ, по-моему, незаслуженно недостаточно распространен среди программистов, хотя дает отличные результаты. Речь идет о программе в машинных кодах "ON ERROR GO TO". Те, у кого есть фирменная программа "SUPERCODE" или ее более поздние версии, возможно знают об упомянутом блоке кодов, но не применяют его из за недостаточного удобства в работе. Методика, изложенная ниже, позволит достаточно легко и эффективно пользоваться этой программой, а для тех, у кого нет программы "SUPERCODE", приводится блок кодов "ON ERROR GO TO". Для тех, кто интересуется программированием в машинных кодах, подробно описывается его работа, а если Вы не интересуетесь машинными кодами, то можете пропустить описание работы блока "ON ERROR GO TO". Прочитав раздел о примере использования этого блока, Вы сможете "пристыковать" его к любой имеющейся у Вас Бейсик-программе. Итак, сначала о блоке кодов "ON ERROR GO TO".

1. Блок кодов "ON ERROR GO TO"

Этот блок имеет длину 73 байта. Он является релоцируемым, то есть его можно загружать в любое место памяти. Вызывается он обычным способом, то есть при помощи RANDOMIZE USR ADDR, где ADDR - это начальный адрес загрузки блока кодов.

Рассмотрим работу блока, рас положив его для примера в буфере принтера начиная с адреса 23296 (#5B00).

В листинге блока кодов справа число в скобках - это ссылки на комментарии по работе блока, описание которой дано ниже.

Текст программы "ON ERROR GO TO"

```
5B00 CD7C00       CALL  #007C        (1)
5B03 3B           DEC   SP           (2)
5B04 3B           DEC   SP
5B05 E1           POP   HL           (3)
5B06 010F00       LD    BC,#000F     (4)
5B09 09           ADD   HL,BC        (5)
5B0A EB           EX    DE,HL        (6)
5B0B 2A3D5C       LD    HL,(#5C3D)   (7)
5B0E 73           LD    (HL),E       (8)
5B0F 33           INC   HL
5B10 72           LD    (HL),D
5B11 C9           RET   (9)
5B1E 3B           DEC   SP           (10)
5B13 3B           DEC   SP
5B14 CD8E02       CALL  #028E        (11)
5B17 7B           LD    A,E          (12)
5B18 FEFF         CP    #FF
5B1A 20F8         JR    NZ,#5B14     (13)
5B1C 3A3A5C       LD    A,(#5C3A)    (14)
5B1F FEFF         CP    #FF
5B21 2821         JR    Z,#5B44      (15)
5B23 FE07         CP    #07          (16)
5B25 2B1D         JR    Z,#5B44
5B2T FE08         CP    #08          (17)
5B39 2819         JR    Z,#5B44
5B2B 3C           INC   A            (18)
```

```
5B2C 32815C      LD    (#5C81),A    (19)
5B2F FD3600FF    LD    (IY+0),#FF   (20)
5B33 210000      LD    HL,#0000     (21)
5B36 22425C      LD    (#5C42),HL   (22)
5B39 AF          XOR   A            (23)
5B3A 32445C      LD    (#5C44),A
5B3D FDCB01FE    SET   7,(IY+1)     (24)
5B41 C37D1B      JP    #1B7D        (25)
5B44 33          INC   SP           (26)
5B45 33          INC   SP
5846 C30313      JP    #1303        (27)
```

Блок "ON ERROR GO TO" состоит из двух частей. Собственно программа по обработке ошибки начинается с адреса #5B12 (23314). А с адреса #5B00 (23296) расположена процедура привязки блока кодов к тем адресам, в которых он находится. Привязка осуществляется следующим образом.

Сначала вызывается подпрограмма из ПЗУ (1). По указанному адресу #007C расположена одна единственная команда: RET. При выполнении инструкции CALL на стек компьютера помещается адрес ячейки, в которой находится следующая за CALL команда, то есть в нашем случае #5B03 (23299), а указатель стека - регистр SP - уменьшает свое значение на два.

Встречая инструкцию RET, выполнение программы продолжится с адреса, взятого со стека - это #5B03, а указатель стека увеличит свое значение на два. Но число #5B03 не стирается из памяти. Чтобы возвратить его, производится уменьшение указателя стека на два (2) и число со стека записывается в регистр HL (3). (При этом указатель стека опять увеличивается на два, то есть возвращается к своему прежнему значению.) Теперь в регистре HL находится число #5B03 (23299).

Далее (4) в регистр BC заносится величина смещения #000F (015) байт и добавляется к содержимому регистра HL (5). Теперь в регистре HL будет #5B03+#000F=#5B12 (23299+15=23314). Это адрес начала программы по обработке ошибки.

Если расположить блок кодов "ON ERROR GO TO" с любого другого адреса, то в результате действий (1)...(5) в регистре HL будет адрес начала программы по обработке ошибки. Далее этот адрес пересылается из регистра HL в регистр DE (6) а в регистр HL загружается (7) двухбайтное число из ячеек #5C3D, #5C3E (23613, 23614). Это адрес системной переменной ERR SP. При ошибке происходит переход на адрес, который содержится на стеке в ячейке, адрес которой находится в ERR SP. До работы программы "ON ERROR GO TO" там был адрес 4867 (#1303) - это в ПЗУ подпрограмма вывода сообщения об ошибке. Этот адрес перехода подменяется на новый, то есть на #5B12 (23314), записываемый на стек побайтно из регистров E и D (8). После этого происходит возврат в Бейсик-программу (9).

На этом подготовительная часть закончена. Если ошибки при работе Бейсик-программы не происходит, то все работает обычным порядком, как и без программы "ON ERROR GO TO". А если происходит ошибка, то компьютер передает управление на адрес, записанный на стеке в паре ячеек, указанных в системной переменной ERR SP, то есть на #5B12. При этом происходит следующее.

Вначале указатель стека уменьшается на два, чтобы не испортить стек при работе блока кодов "ON ERROR GO TO" (10). Затем вызывается подпрограмма из ПЗУ "KEY-SCAN" (11). Результат действия этой подпрограммы отражается в регистре DE. Точнее, в регистре E - номер нажатой клавиши. Если не нажато ни одной клавиши, то в регистре E - #FF (255) (аналогично оператору IN в Бейсике).

Сравнивая содержимое регистра E с #FF, предварительно переслав его (12) в аккумулятор A, определяется факт нажатия на какую-либо клавишу. Если результат сравнения не ноль, то есть нажата какая-нибудь клавиша (это может быть, например, клавиша "BREAK"), то зацикливается подпрограмма "KEY-SCAN", приостанавливая дальнейшие действия. Как только клавиша будет отпущена, в регистре E появится число #FF и продолжится работа программы "ON ERROR GO TO"

Далее анализируется произошедшая ошибка, путем проверки содержимого ячейки системной переменной ERR NR, расположенной по адресу #5C3A (23610). Это код ошибки минус 1. Для этого содержимое ERR NR пересылается в аккумулятор (14). Если нет ошибки, а это может произойти в случае логичного завершения программы и ее запланированной остановки в конце (0 OK), то есть код ошибки равен нулю, то содержимое ячейки 23610 будет 0-1 (что для одного байта эквивалентно 256-1) то есть 255 (#FF). В этом случае управление будет передано (15) на адрес #5B44 (23364). Здесь (26) возвращается прежнее значение указателю стека SP и осуществляется переход на подпрограмму вывода сообщения об ошибке (27) (в нашем случае это будет сообщение 0 OK), как это было бы при обычной работе Бейсик-программы.

Аналогичные действия будут и в случае появления ошибки с кодом 8 (8 END of file), (то есть PEEK 23610=7), такая ошибка может произойти при работе с внешней памятью, а также в случае появления ошибки с кодом 9 (9 STOP statement), (то есть PEEK 23610=8), если в тексте программы стоит оператор STOP. Сравнивается содержимое аккумулятора с числом 7 и с числом 8 и в случае совпадения, управление передается (16), (17) на адрес #5B44 (23364).

Дальнейшие действия являются подготовкой к запуску Бейсик-программы с заданной строки. Они необходимы для корректной дальнейшей работы интерпретатора Бейсика.

Для этого прежде всего в ячейку, отведенную для системной переменной ERR NR, заносится число #FF, имитируя отсутствие ошибки (20). Здесь следует пояснить, что при работе интерпретатора Бейсика в "Спектруме" в регистровой паре IY находится число #5C3A (23610) - адрес системной переменной ERR NR.

Далее в два байта системной переменной NEW PPC (ее адрес - #5C48) записывается номер Бейсик-строки, на которую должен быть сделан переход. Для того, чтобы осуществить это, необходимо предварительно номер строки для перехода занести (21) в регистр HL и затем (22) содержимое HL переслать по адресу #5C48. Здесь отметим для себя, что номер Бейсик строки для перехода находится в ячейках #5B34, #5B35 (23348-23349). К этому моменту мы еще вернемся позже.

Затем обнуляется ячейка системной переменной NS PPC - это номер оператора в строке, на которую будет сделан переход. Сначала обнуляется аккумулятор A (23) и затем содержимое A пересылается в NS PPC - ячейку #5C44 (23620).

Теперь надо установить флаг синтаксиса: 7-й разряд системной переменной FLAGS - управляющие флаги Бейсика, имитируя положительный результат проверки Бейсик-строки на синтаксис (24). Адрес FLAGS - #5C3B (23611) (или IY+1).

Теперь подготовка Бейсик-системы закончена и можно запускать интерпретатор Бейсика (25). Таким образом, Бейсик-программа будет запущена со строки, номер которой задан двухбайтным числом по адресу #5B34 (23348).

Некоторые моменты в работе блока кодов могут показаться излишними, но благодаря им блок "ON ERROR GO TO" устойчиво и надежно работает в разных режимах.

Для того, чтобы получить блок кодов "ON ERROR GO TO", наберите следующую Бейсик-программу:

```
10 LET n=23296: LET s=0
20 FOR x=n TO n+72
30 READ y
40 POKE x,y
50 LET s=s+y
60 NEXT x
70 IF s<>7298 THEN PRINT FLASH 1;"ERROR": STOP
80 SAVE "on err" CODE 23296,73
100 DATA 205,124,0,59,59,255,1,15,0,9,235,42,61,92,115,35,114,201
110 DATA 59,59,205,142,2,123,254,255,32,248,58,58,92,254,255,40,33,254,7,40,29,254,8,40,25
120 DATA 60,50,129,92,253,54,0,255,33,0,0,34,66,92,175,50,68,92,253,203,1,254
130 DATA 195,125,27,51,51,195,3,19
```

После старта, если Вы все набрали правильно, программа сформирует блок кодов и

выдаст его для записи на магнитофон.

Для того, чтобы показать работу программы "ON ERROR GO TO", нам понадобится демонстрационная Бейсик-программа. Это будет программа под названием "BEEPER". Задача, решаемая ей, примитивна и не представляет практического интереса, но на этом примере будут продемонстрированы возможности программы "ON ERROR GO TO" и методы ее практического использования.

2. Программа "BEEPER"

Текст программы:

```
1 GO TO 100
2 BORDER 7: PAPER 7: INK 0: CLS
3 LOAD "on err" CODE 23296
4 RUN
5 SAVE "BEEPER" LINE 2
6 SAVE "On err" CODE 23296,73
7 STOP
20 INPUT ;
22 PRINT #0; TAB 8; FLASH 1;"press any key"
24 PAUSE 0
26 RETURN
100 BORDER 1: PAPER 7: INK 0: CLS
110 PRINT AT 12,8; PAPER 2; INK 7; BRIGHT 1;"  B E E P E R  "
120 GO SUB 20
130 BORDER 7: CLS: PRINT AT 21,0;
1000 REM "beeper"
1010 INPUT "Enter tune (-60...69): ",t
1020 PRINT "Tune=";t
1030 PRINT #0;TAB 8;INVERSE 1;"    B E E P    "
1040 FOR n=1 TO 50
1050 BEEP .07,t
1060 NEXT n
1070 GO SUB 20
1080 GO TO 1000
```

После того, как Вы наберете программу, запустите ее со второй строки RUN 2 и загрузите блок кодов "on err" CODE. После окончания загрузки нажмите "BREAK" и выгрузите готовую программу на ленту, выполнив RUN 5.

Теперь можете командой RUN запустить программу сначала. После появления "заставки", нажмите любую клавишу. В ответ на запрос о величине тона введите число от -60 до 69 и нажмите любую клавишу. Вы услышите прерывистый звук, продолжающийся около пяти секунд, затем программа зацикливается, запрашивая следующую величину тона.

Теперь несколько слов о ситуациях, при которых происходит остановка программы.

Остановка с сообщением "L BREAK into program" произойдет при нажатии на клавишу "BREAK" в режиме ожидания, когда появляется мигающая надпись "press any key" или в момент звукового сигнала. Остановка с сообщением "D BREAK - CONT repeats" произойдет, если нажать "BREAK" в момент ожидания загрузки блока кодов (строка 2). Если при загрузке блока кодов произойдет ошибка магнитофона, то программа остановится с сообщением "R tape loading error". Программу можно остановить, если на запрос о величине тона вместо числа ввести букву, например a,b,... В этом случае остановка произойдет с сообщением "2 Variable not found". Если введенная величина тона будет ниже 60 или выше 69, то программа остановится при попытке выполнить оператор BEEP с сообщением "B integer out of range".

Чтобы запустить программу "ON ERROR GO TO", изменим строку 1:

```
1 RANDOMIZE USR 23296 GO TO 100
```

Теперь при любой ошибке, произошедшей после команды RUN, программа будет запускаться со строки, номер которой задан двухбайтным числом по адресу 23348. Так как там 0, то программа перезапустится с начальной строки (в нашем случае с первой строки). Теперь остановить программу невозможно, Вы можете убедиться в том, что при любой попытке остановки программа просто перезапустится заново.

Это не совсем тот эффект, который нам необходим, но на этом этапе становится ясно, что остановить программу после старта невозможно, можно только перезагрузить ее с ленты, а это неудобно при отладке программы. Поэтому надо позаботиться о возможности остановки программы в процессе отладки. Это можно сделать, например, введя такой "жучок":

```
25 IF INKEY$ "q" THEN STOP
```

Теперь, когда на экране появляется мигающая табличка "press any key", то программу можно остановить, нажав клавишу "Q".

Для того, чтобы инициировать программу "ON ERROR GO TO ", надо, чтобы было выполнено: RANDOMIZE USR 23296. Эту команду можно подставить в начало строки, с которой происходит запуск программы (строка 1) Далее, на разных этапах выполнения программы надо изменять содержимое ячеек 23348,23349, чтобы управлять переходом по ошибке согласно задуманной логике. Например, добавим строки:

```
115 POKE 23348,24 POKE 23349, 0
1000 POKE 23348.242 POKE 23349.3
1065 POKE 23348.0 POKE 23349,0
```

Строка 115 блокирует клавишу "BREAK", исключая остановку программы в режиме "заставки", возвращая ее все время на строку 24 - ожидание нажатия какой-нибудь клавиши. Строка 1000 определяет возврат на строку 1010 при ошибке ввода данных (PEEK 23348+256*PEEK 23349=1010). Строка 1065 определяет возможность перезапуска программы сначала при нажатии клавиши "BREAK", когда появляется мигающая надпись "press any key".

Теперь можно считать, что мы добились требуемой логики работы программы. Предотвращается остановка программы при ошибке ввода данных. Кроме того, можно прервать звуковой сигнал в момент исполнения (когда появляется табличка "BEEP"). А когда исполнение закончится можно вообще перезапустить программу, нажав "BREAK".

Однако следует отметить еще один момент. Определенное неудобство доставляет перевод номера строки в двухбайтную форму. Это приходится делать вручную. К тому же если придется переделывать программу в процессе отладки, изменяя нумерацию строк, то опять приходится пересчитывать данные для POKE 23348, POKE 23349.

Задачу можно значительно упростить если вспомнить о том, что двухбайтный конвертер в "Спектруме" уже есть. Это оператор RANDOMIZE, который задает начальное значение для функции случайной величины RND. Подав команду RANDOMIZE n, мы устанавливаем системную переменную SEED, используемую для вычисления очередного значения функции RND. SEED расположена по адресу 23670 и занимает два байта. В ячейке 23670 находятся младший, а в ячейке 23671 - старший байты числа n, которое используется с оператором RANDOMIZE. Подайте команду RANDOMIZE 1010. Теперь сделайте:

```
PRINT PEEK 23670
PRINT PEEK 23671
```

- получите 242 и 3

Теперь усовершенствуем нашу программу, добавив строки:

```
10 POKE 23348, PEEK 23670: POKE 23349, PEEK 23671: RETURN
```

Изменим строки

```
115 RANDOMIZE 24: GO SUB 10
1000 RANDOMIZE 1010: GO SUB 10
1065 RANDOMIZE 1: GO SUB 10
```

Теперь, с точки зрения программирования, задача упростилась. Здесь только надо отметить, что нельзя подавать команду RANDOMIZE 0, так как в этом случае переменная SEED принимает не значение 0, а становится равной другой системной переменной FRAMES - счетчика кадров, обеспечивая практически случайное число. Вместо RANDOMIZE 0 можно подавать RANDOMIZE 1, так как программа все равно обычно начинается со строки 1. (Если же у Вас присутствует нулевая строка и Вы хотите стартовать именно с нее, то просто подставьте в требуемое место, как и раньше, POKE 23348,0: POKE 23349, 0)

Сейчас можно удалить строку 25, которая нужна была для отладки программы. Теперь единственным путем, позволяющим остановить программу, является нажатие "BREAK" при загрузке с ленты, в тот момент, когда Бейсик-программа уже загружена и ожидается ввод блока кодов "on err" (строка 2).

При желании можно ликвидировать и эту возможность. Для этого блок кодов "ON ERROR GO TO" можно расположить внутри Бейсик-программы в нулевой строке. Наберите:

```
1 REM
```

а после REM - 73 пробела или любых других символов. Затем сделайте POKE 23756,0. Первая строка стала нулевой. Теперь выполните LOAD "on err" CODE 23760.

Затем измените строки:

```
2 RANDOMIZE 1: GO SUB 10: GO TO 100
10 POKE 23812,PEEK 23670: POKE 23813, PEEK 23671: RANDOMIZE USR 23760 RETURN
```

Теперь, когда блок "ON ERROR GO TO" расположен в новом месте (начиная с адреса 23760), ячейками в которых задана строка для перехода по ошибке, будут 23812, 23813.

Строки 1, 3, 6 - удалите, они больше не нужны, так как теперь наша программа состоит из одного блока вместо двух и "горячий" старт (RUN 2) совпадает с "холодным" стартом (RUN). В этом варианте изменен также способ инициирования блока кодов "ON ERROR GO TO". Команда RANDOMIZE USR 23760 помещена теперь не в первую (стартовую) строку, как раньше, а в строку 10. Поэтому включение блока в работу происходит теперь в любом случае, когда выполняется команда GO SUB 10, независимо от того, с какой строки Вы запустите Вашу программу. Такой вариант более универсален.

Мы рассмотрели работу блока кодов "ON ERROR GO TO" на примере маленькой программы. В больших программах может потребоваться более сложная логика перехода при ошибке. Для этого используйте ячейку системных переменных 23681, куда заносится код ошибки при работе блока "ON ERROR GO TO". Анализируя PEEK 23681, Вы можете организовать переход на нужную строку. Для примера измените строку 1000 программы "BEEPER".

```
1000 RANDOMIZE 2000 GO SUB 10
```

Добавьте строки:

```
2000 IF PEEK 23681=11 THEN INPUT ;: PRINT #0; PAPER 2; INK 7; BRIGHT 1; " W A R N I N G :
     60<Tune<69  ": BEEP 1,0: PAUSE 100
2010 GO TO 1010
```

Теперь в том случае, если вводимая величина будет выходить за пределы –60…+69, появится предупредительная табличка со звуковым сигналом. Конечно, проверку на допустимые пределы логичнее организовать при помощи обычного Бейсика, просто этот пример показывает, как можно использовать код ошибки. Хотя надо сказать, что в своих программах мне еще ни разу не приходилось этим приемом пользоваться.

В заключение хочу сказать, что при отладке программ с блоком "ON ERROR GO TO" почаще делайте RUN 5 (сохранение программы на ленте). В результате ошибочных действий может случиться так, что Вы не сможете остановить программу. Тогда останется только загрузить предыдущий вариант. Если предполагаются значительные изменения в программе, то временно в начало строки 10 подставьте RETURN: это отключит блок "ON ERROR GO TO". Кроме того всегда предусматривайте "жучок" для возможности остановки программы типа строки 27 в программе "BEEPER".

* * *

В следующий раз мы поговорим о структуре Бейсик программ, о том, с чего практически начинать написание новой программы, то есть когда в уме или на бумаге план уже достаточно "созрел" и Вы включили компьютер, чтобы набивать текст программы, а также о том, как облегчить себе жизнь, предусмотрев элементарный сервис для себя.

FORUM

В 11-12 номере "ZX-РЕВЮ" (стр. 254) за прошлый год мы писали о существовании "циклических" защит программ от копирования и упоминали защиты класса ALKATRAZ LOADER.

В ответ на эту заметку пришло несколько писем. Победы, одержанные нашими читателями над этим загрузчиком с помощью аппаратных средств мы рассматривать не будем, считая этот путь не относящимся к теме, а вот частный случай, с которым разобрался Д.Коронцвит из Г. Жуковского, Моск. обл., мы рассмотрим.

Почему нас не интересуют аппаратные пути взлома и копирования программ? Дело в том, что с программистской точки зрения взлом ради взлома, копирование ради копирования не интересны. Нас интересуют программистские приемы, новые методы, короче интересует все то, на чем можно учиться и набирать опыт. И, если быть до конца честными, то надо признаться, что любые статьи, посвященные вскрытию программ и снятию защит в основном служат не тем, кто их снимает, а тем, кто их ставит и тем, кто учится программировать.

Итак, по порядку. Просматривая загрузчики, написанные Сергеем Скоробогатовым для дискофицированных им программ Winter Edition, Chase H.Q., Agent X и др., наш читатель столкнулся с листингом, который выглядел примерно так (см. листинг 1, комментарий к листингу - наш, "ИНФОРКОМ"):

Декодирование вручную, с помощью MONS3 показало, что то, что было АБРАКАДАБРОЙ, содержит блок, очень похожий на то, что Вы видите на листинге. Изменился только "ключ" и конечно изменился адрес, загружаемый в регистровую пару HL. Дальнейшее декодирование открыло еще один аналогичный блок и т.д. Длина и структура всех трех просмотренных блоков были одинаковыми.

Конечно, это лишь частный случай, но этот факт позволил написать программу, которая все последующее декодирование выполнила автоматически. Конечно, в более общих случаях она не сработает, но сам принцип будет полезен начинающим, а по мере необходимости они смогут адаптировать метод к конкретным условиям.

Пример декодирующей процедуры приведен в листинге 2.

ЛИСТИНГ 1

```
        LD HL,nn     - загрузили адрес, с которого начинается декодируемый блок.
        LD BC,nn     - длина этого блока (организовали счетчик).
 LOOP   LD A,(HL)    - приняли байт для декодирования.
        XOR "ключ"   - само декодирование.
        LD (HL),A    - заслали декодированное значение на место АБРАКАДАБРЫ.
        INC HL       - перешли к очередному байту.
        DEC BC       - уменьшили счетчик байтов на единицу.
        LD A,B       - подготовка к проверке счетчика на ноль.
        OR C         - проверка счетчика на ноль.
        JR NZ,LOOP
        ..................
        АБРАКАДАБРА
        ..................
```

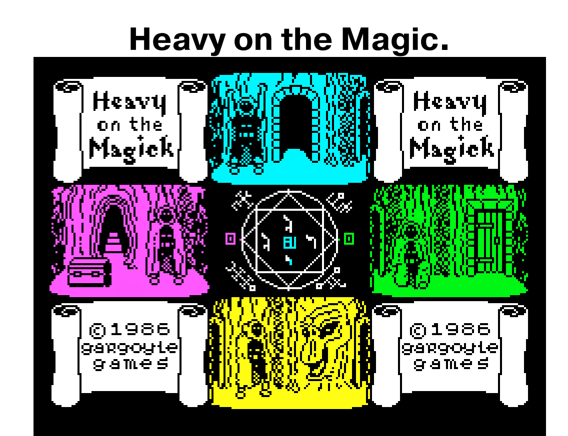
ЛИСТИНГ 2

```
AGAIN   LD IX (aa)      -   aa - адрес ячейки, в которой организовано хранение адреса начала
                            "взламываемого" блока.
        LD L,(IX+1)     -   второй и третий байты исследуемого блока
        LD H,(IX+2)         содержат адрес декодируемого куска
        LD C,(IX+4)     -   пятый и шестой байты исследуемого блока
        LD D,(IX+5)         содержат длину декодируемого куска.
LOOP    LD A,(HL)       -   приняли байт для декодирования.
        XOR (IX+8)      -   само декодирование.
        LD (HL),A       -   заслали декодированное значение на место.
        INC HL          -   перешли к очередному байту.
```

```
DEC BC          -   уменьшили счетчик байтов на единицу
LD A,B          -   подготовка к проверке счетчика на ноль.
OR C            -   проверка счетчика на ноль.
JR NZ,LOOP      -   если не все биты декодированы, переход к декодированию очередного
                    байта.
LD BC,010H      -   длина декодирующего блоке - 16 байтов (010H).
ADD IX,BC       -   "перепрыгнув" через рассмотренный блок, вводим в IX адрес начала
                    следующего декодирующего блока.
LD (aa),IX      -   помещаем его адрес в ячейку с адресом aa.
LD A,0EEH       -   0EEH - это (238 в десятиричной системе) код операции XOR.
CP (IX+7)       -   сравниваем содержимое восьмого байта нового декодирующего блока с
                    кодом 238, ожидая, что там будет стоять XOR.
JR Z,AGAIN      -   если это так, то возврат к началу и декодирование следующего блока.
RET             -   если нет, то возврат в вызывающую программу. Отсутствие XOR в данной
                    позиции может говорить либо о том, что все декодирование успешно
                    завершено, либо о том, что защита сменила принцип кодирования и уже
                    не использует XOR или использует не только XOR. Тогда надо
                    остановиться и посмотреть, что же она использует и по-возможности
                    внести изменения в свою декодирующую процедуру.
```

Адвентюрные игры.

Heavy on the Magic.



Свое исследование этой увлекательной игры, выпущенной фирмой GARGOYLE GAMES в 1986 году прислал наш читатель из Грозного Каракашев А.Г. С его полезными советами, посвященными игре Sceptre of Bagdad Вы знакомы по прошлому выпуску ZX-РЕВЮ.

Ему удалось пройти все игровое программное пространство до конца, он посетил все комнаты замка, но не может считать, что разобрался с программой полностью. Есть еще нерешенные проблемы, может быть кто-то знает подходы и к ним? Наш корреспондент пользовался некоторыми ходящими по рукам непроверенными описаниями и ряд вопросов связаны именно с ними.

Начнем с проблем, а потом перейдем к достижениям.

В описаниях говорится о 21 типе монстров. Обойдя весь замок, насчитать такого количества не удалось, даже если к монстрам отнести демонов, огонь, воду и персонажей, неперемещающихся в пространстве и не нападающих на главного героя.

В описаниях говорится о 280 предметах, которые можно исследовать. Если считать все, включая двери, столы, камни, сталактиты, никак более 250 не получается.

В описаниях говорится о том, что надписи на стенах встречаются довольно часто, удалось увидеть только одну - в комнате слева от начальной. Более того, нет нигде обещанной комнаты со множеством дверей, где висит особый знак, прочитав который можно погибнуть.

Вскрыв игру с помощью COPY-COPY, удалось обнаружить следующие имена демонов: ASTABOT, ASMODEE, BELEZBAR и MAGOT, но ни в каких книгах заклятий о них не упоминается.

Таким же путей взлома удалось найти слова: ADEPTUS MAJOR, ADEPTUS MINOR, MAGISTR TEMPLI, MAGUS. И невооруженным глазом видно, что эти слова обозначают магический ранг. Но нигде эти ранги не участвуют. Открыв все двери и обойдя все, что

можно, герой повысил свой ранг от NEOPHYTE до PHILOSOPHUS через ZELATOR и PRACTICUS. Причем последняя дверь открывается только при ранге PHILOSOPHUS - не больше, не меньше.

Кто знает, где и как можно получить те ранги, которые были обнаружены при вскрытии программы? Отзовитесь! Может быть у кого-то есть фирменное описание игры?

Теперь несколько заметок для тех, кто еще не начал работать с этой увлекательной программой.

На самом первом экране Вы видите Аксила между двумя столами с книгами. Левая - отравлена, а вот правая - GRIMOIR ("Гримуар") содержит заклятья: BLAST, INVOKE, FREEZE.

С помощью заклятья BLAST можно уничтожать монстров, населяющих замок. Впрочем, для борьбы с самыми крепкими из них, придется прибегать еще и к помощи специальных предметов.

К дверям, около которых есть столы, надо подбирать ключи. Когда Аксил кладет на стол нужный ключ, дверь открывается.

К тем дверям, около которых есть знак в виде двух переплетенных колец, необходимо кроме ключа принести еще и мешочек с золотом (BAG OF GOLD) и положить его на стол. Мешочки с золотом взаимозаменяемы, т.е. любой такой мешочек (кроме отравленного) подойдет к любой такой двери.

Двери, возле которых есть охрана, открываются паролями.

Проход в следующие три двери повышает магический ранг:

ASK APEX - пароль "DOOR, SI LENCE" (комната ZELATOR).

SEEK FIRE BIRD TO ENTER DOOR - пароль "DOOR, LAZA" (комната PRACTICUS).

THE GREAT SIGN I IN FREE - пароль "DOOR, SORONOROS" (комната PHILOSOPHUS).

Прочие двери:

CRY AMD ENTER DOOR - пароль "DOOR, WOLF" - ведет на первый этаж.

TO ENTER IS MADNESS - пароль "DOOR, LUNACY" - открывает дальнейшие глубины замка.

SAY NUMBER OF MAGIC WORDS -пароль "DOOR, ELEVEN" - назначение неясно. Пройдя эту дверь, Аксил исполняет какой-то победный танец и получает сообщение, внушающее уверенность в собственных силах: Well done! Axil the able you can made it for an exit.

Аналогична ей и дверь: EYE FOR AN EYE TO ENTER PARADISE - она открывается паролем "DOOR, LONG".

Интересна дверь PILE TOMB NOKEY. Для того, чтобы ее открыть, надо привлечь на помощь демона: INVOKE ASMODEE-, а затем дать ему команду "ASMODEE, DOOR", - и он разрушит дверь.

Теперь несколько слов о демонах. Каждому демону соответствует свой талисман. Поэтому, чтобы вызвать демона, надо выложить этот талисман в комнате. Если его вызвать без талисмана, он жестоко наказывает, отправляя героя в адские печи.

Для демона BELESBAR нужен талисман MANTIS. Функции этого демона пока неясны, единственное, что удалось установить - он дублирует команду EXEMINE.

Демон MAGOT предпочитает SUNFLOWER. Он знает, где в замке расположен какой-либо предмет.

Предмет демона ASTAROT - меч (SWORD). Кажется, это самый полезный из демонов. Он выполняет роль телепорта и перенесет Вас в названную Вами область замка. Правда, есть недостаток - за мечом приходится возвращаться пешком, поскольку второй раз его уже не вызвать.

ASMODEE - самый злобный демон. Его можно вызвать только в комнате, в которой лежит рубин (RUBY). Он реагирует только на команду DOOR и разрушает дверь, если Вы имеете ранг PHILOSOPHUS, в противном случае с ним лучше дела не иметь.

Вызывая демона, Вы должны помнить, что он появляется точно над талисманом, поэтому лучше встать от него чуть подальше.

Теперь ряд полезных советов.

1. С помощью CLASP можно пройти через огонь.

2. NOUGAT можно положить на место NUGGET.

3. С помощью NUGGET можно убить оборотня (WEREWOLF).

4. С помощью зеркала (MIRROR) можно победить Медузу (MEDUSA).

5. Вампира побеждают чесноком (GARLIC).

6. SLAT побеждает циклопа.

7. BALL можно положить на место PELLET.

8. С помощью PELLET можно победить SLUG.

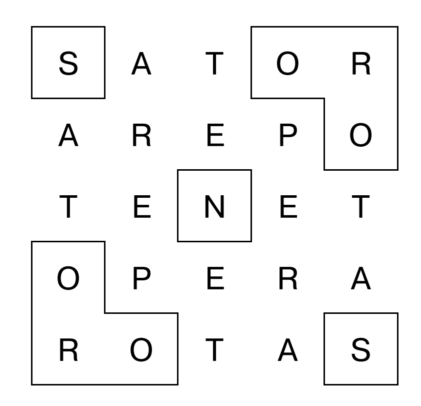
9. Вместо яйца (EGG) положить скорлупу (SHELL).

10. Яйцо используется следующим образом. В районе NIDUS на первом уровне замка, где живут циклопы, нужно положить яйцо в огонь и вызвать птицу-феникс "NEST,PHOENIX". Феникс должен Вам сообщить пароль LAZA.

11. Гидру (HYDRA) можно пройти с помощью SNAKE.

12. Компоненты ULNA, HEAD, THIGH нужно сложить в котел и произнести заклятие "CAULDRON, ACHAD". В котле возникает новый монстр: AU, холодный и мертвый. Он подсказывает пароль LAZA, который уже сообщал Феникс. Может быть это сбой в программе, связанный с неудачным снятием ее защиты? Может быть. AU должен сказать что-то другое, например пароль LONG, который был извлечен из программы просмотром COPY-COPY и подбором.

13. Так же, просмотром кода был получен и пароль SORONOROS, хотя можно предположить, что его можно было бы вывести и из надписи на стене:



14. Чтобы пройти через воду, служит заклятие "WATER, FALL".

15. Пройти через пропасть (CHASM) можно, имея FLASK.

16. В одной комнате двери не оказалось. Трое стражников сообщили, что южной двери не будет, пока не найдешь пароль с помощью ERLSTONE. Спросив об этом у MAROMа, получаешь ясный ответ, что искать его надо в районе PIT: Seek it in the pit. PIT находится в замке на четвертом этаже, но никакого ERLSTOME там нет. Демоны APEX и BELEZBAR на вопрос об этом просят показать им ERLSTONE.

Однако, с помощью хитрости, пройти за несуществующую дверь все же удалось. Просмотр в COPY-COPY дал слово LICHGATE, которое до этого нигде не использовалось. Сказав "ASTAROT, LICHGATE" в комнате с мечом, удалось попасть за эту дверь, так и не найдя ERLSTONE.

Таким образом, был пройден весь замок: 4 этажа, 8 X 8 комнат, но нет никакого выхода и нет естественного финала. Может быть, кто-то из читателей прошел игру до конца и сможет указать на ошибку в действиях, а может быть это сбой в программе, связанный с неудачным снятием защиты?

И еще: нигде не удалось использовать кость (RIB) и кости (BONES - три штуки).

В заключение наш читатель просит начать мозговой штурм игры "DUN DARACH", выпущенной той же фирмой GARGOYLE GAMES. "ИНФОРКОМ" присоединяется к этому пожеланию и обращает внимание на то, что есть еще две игры: TIR-NA-NOG и MARSPORT, не менее интересные и тоже пока никем не освещенные.

Полезные советы.

В прошлом году мы несколько раз писали об особенностях работы компьютеров "Дубна 48" (с. 115, 156). Мы просили тех читателей, которые найдут программные пути

повышения совместимости этой популярной модели, поделиться своими достижениями со всеми любителями "Спектрума".

Суть проблемы, если Вы помните, состоит в том, что из-за нехватки быстродействующих микросхем памяти в этой модели был применен процессор с пониженной частотой, для чего были изменены некоторые константы в процедурах ПЗУ, управляющих загрузкой программ. В итоге программы, снабженные спецзагрузчиками, обходящими ПЗУ, могут не загружаться.

Своим видением этой небольшой проблемы и своими подходами сегодня делится наш читатель из поселка Провидения Магаданской обл. Михаил Владимирович Лапырев.

Во-первых, есть копировщики, работающие на "Дубне", которые позволяют Вам откопировать программу.

1. Lady COPY.
2. COPY FM3
3. COPY-COPY (Pirate 02)

Эти копировщики, правда, не компрессирующие, но свою задачу выполняют хорошо.

И будет совсем прекрасно, если Вы достанете турбо-копировщики:

1 TURBO-COPY.
2 TURBO-CAC.

Эти копировщики - компрессирующие и на "Дубне" они работают, хотя надо знать один нюанс. После загрузки копировщика и выхода в стартовое окно, не спешите нажимать LOAD. Включите магнитофон с программой, которую Вы хотите переписать и дождитесь пилоттона, идущего перед блоком программы, и только потом нажимайте клавишу. Не беспокойтесь, если между блоками копировщик будет принимать посторонний сигнал и выдавать сообщение об ошибке. После начала следующего пилотсигнала снова нажмите LOAD и так далее. Всю загрузку в копировщик выполняйте в режиме "T".

А вот с выгрузкой придется немного помудрить. Если программа стандартная, то ее можно выгружать, как обычно, клавишей "A" (АВТО) в режиме "T". Если же у программы свой загрузчик, то первый блок БЕЙСИК-загрузчика и загрузчик машинного кода надо выгружать в режиме "T", а остальные блоки - в режиме "L". После такой переделки программа будет работать на компьютере "Дубна". Правда, возрастет примерно в 2 раза время загрузки и увеличится расход ленты. Надо также помнить, что переделанные таким образом программы уже не смогут работать на других машинах.

Таким приемом удалось адаптировать под "Дубну" многие из первоначально неработавших программ. Конечно, не все еще доведено до конца. Например, не поддается переделке программа SPOOKED, у которой нестандартный загрузчик выполняет как бы загрузку "с конца блока". Но это поле для новых экспериментов.

Советы и секреты.

CHRONOS - если в таблице результатов набрать YING IT BABY (причем обязательно заглавными буквами), то получите бесконечную энергию.

COBRA FORCE - если переназначить клавиши в игре на "SIMON", Вы получите бесконечную жизнь.

GEMINI WINGS - пароли:
уровень 2 - EYEPLANT
уровень 3 - WHATWALL
уровень 4 - GOODNITE
уровень 5 - SCULLDUG
уровень 6 - WIGMOUTH
уровень 7 - GREEPISH

PIPEMANIA - некоторые пароли:
уровень 5 - DISK

уровень 9 - NAIL
уровень 13 - ONCE
уровень 17 - ROPE
уровень 21 - PENS
уровень 25 - SLIP
уровень 29 - EACH
уровень 33 - RISE

AFTERBURNER: POKE 37934,0: 37935,0: 37936,0

RET STORM - POKE 37337,201

Секретами поделился Фильков Андрей Павлович из Москвы.

Письмо читателя.

Здравствуйте, "ИНФОРКОМ". Я прочитал Ваш трехтомник по изучению и работе с машинным кодом. Пока я не ознакомился с вашим изданием, я без успеха натыкался на барьер машинного кода. Мне всего тринадцать лет, но даже мне было нетрудно вникнуть в особенности процессора Z-80.

Я являюсь подписчиком "ZX РЕВЮ" на 91-92 годы. До того, как я не прочитал вашего издания, рубрику "Машинные коды" я просто пролистывал, уделяя ей ноль внимания.

Большое спасибо от всех людей, которые пользуются ПЭВМ "ZX-Spectrum".

Коллекция игр у меня небольшая и я очень люблю умные игры, такие как SHERLOCK, KNYGHT TYME и THE HOBBIT. Я очень хотел бы обратиться к экспертам, чтобы они замолвили слово об игре DUN DARACH. Сам же имею по ней очень мало информации.

P.S. Я прошу Вас напечатать в "ZX-РЕВЮ" немного информации.

Существует издание BEEP-SHOW, которое ежемесячно будет ко всем приходить, если Вы обратитесь по адресу: 416510, Астраханская об., г. Ахтубинск-6, Жуковского 24-63, UNICAL-STUDIO.

Этот журнал рассказывает о разных звуковых эффектах. Молчащая программа заговорит и запоет.

Король Юра, г. Свердловск.

* * *

Проблемы совместимости.

Нашим постоянным читателям знакомы работы Полубарьева С.В., направленные на повышение совместимости отечественных моделей Синклер-совместимых машин с их английским прародителем "ZX-Spectrum". "ИНФОРКОМ" традиционно рассматривает проблемы совместимости, как одни из самых основных и сегодня мы предлагаем Вашему вниманию две статьи, посвященные версиям "Ленинград-1" и "Пентагон-48".

Доработки портов ввода/вывода в Sinclair ZX-Spectrum модели "Ленинград-1" (версия Зонова).

ZX-Spectrum-совместимый компьютер "Ленинград-1" (версия Зонова) создавался, по всей видимости, как максимально простой и дешевый вариант машины (или как самый доступный по элементам), о чем свидетельствуют многие примененные в нем схемотехнические решения (совмещенное поле ОЗУ 48 Кбайт и многие другие). Вероятно, именно по этой причине, логика адресации портов ввода/вывода в данной модели упрощена до такой степени, что это вызывает во многих игровых программах малоприятные эффекты мерцания бордюрной рамки экрана в такт с музыкой или звуком выстрела, а иногда и полную несовместимость, выражающуюся в "зависании" программ сразу после загрузки, или же в невозможности управления от KEMPSTON-джойстика. Однако, путем незначительных доработок схемы, эти недостатки можно полностью устранить.

Чтобы Вам полностью понять смысл вводимых изменений, здесь приводится краткая справка об устройстве аналогичных портов I/O фирменного ZX-Spectrum 48K, совместимости с которым мы добиваемся:

В фирменном компьютере для работы с периферийными устройствами используются следующие адреса:

254 (FE HEX)

- по вводу: клавиатура, ввод с магнитной ленты (далее МЛ);

- по выводу: цвет бордюрной рамки дисплея, звуковые эффекты, запись на МЛ.

251 (FB HEX)

- обслуживание фирменного узкопечатного устройства ZX-printer или аналогичных ему Timex-2040 Alphacom-32, Seicosha GP-50S.

31 (1F HEX)

- интерфейс джойстика типа Kempston.

При этом имеются следующие особенности:

1) При работе с портом 254 анализируется только сигнал на линии A0 адресной шины процессора. Поэтому с тем же самым успехом можно обращаться и к любому другому четному порту (252, 250, ... 0), поскольку при его выборке анализируется лишь разряд A0.

2) KEMPSTON-джойстик активируется в том случае, когда в адресе считываемого порта A5="0" и A0="1". При этом на шину данных передается байт, младшие 5 бит которого зависят от положения рукоятки и кнопки "FIRE" джойстика, а старшие 3 бита всегда равны "0". Таким образом, чтобы определить состояние джойстика типа KEMPSTON, программа может обратиться к любому порту с нечетным адресом в диапазоне 1...31.

3) Если ZX-Printer отсутствует в составе системы, то с порта 251 (#FB) должен считываться байт 255 (#FF). В противном случае компьютер будет неизбежно "зависать" при попытке выполнения операторов Бейсика LPRINT и LLIST, а также при загрузке некоторых игровых программ.

Теперь рассмотрим, как организована система ввода/вывода в версии "Ленинград-1":

1) По чтению порта 254 - совместимость с оригиналом полная, но запись в порт 254 выполняется при обращении к любому, а не только "четному" адресу порта (в этом причина мерцания бордюрной рамки). Причина - при выборке ИМС К555ТМ9, на которой выполнен порт вывода 254, адрес вообще не учитывается и не проверяется.

2) KEMPSTON-джойстик имеет сразу 2 дефекта: во-первых, считывается вообще по любому "нечетному" адресу, во-вторых, 3 старших бита установлены в "1" вместо "0".

3) Вытекает из предыдущего пункта: поскольку джойстик считывается по любым, в том числе и не "своим" адресам, то из порта 251 может иногда читаться не совсем то, что надо бы. В авторском варианте схемы это проходит незамеченным, поскольку "1" в старших битах означают отсутствие принтера в системе, но стоит только "занулить" их, и периодические "зависания" машины Вам надежно обеспечены.

Если Вы, прочитав вышеизложенное, узнали описание своих "бед", можете приступить к доработке. Остальным предлагаем убедиться в этом:

- выполните команду PRINT IN 31. На фирменной машине должен быть получен результат 0 (00000000 в двоичном коде). Вы, скорее всего, получите 224 или 255.

- теперь выполните PRINT IN 33 и PRINT IN 251. Если результат первой команды совпадает с предыдущим - нарушена адресация джойстика. Если то же относится и ко второй команде, да ПРИ этом еще и печатается не 255 - Вы уже наверное сталкивались, или скоро столкнетесь с проблемой загрузки игр, которые у Ваших приятелей работали нормально.

- выполните OUT 255,0 и OUT 255, 255. Если рамка экрана стала при этом менять цвет - и здесь нарушена адресация.

К счастью, разработчики платы этой машины предусмотрели в ее углу довольно большое макетное поле - "слепыш", на котором можно разместить все детали, необходимые для доработки.

Вам потребуются ИМС К555ЛЛ1 и К555ЛИ1 (по 1 шт. каждого типа) и некоторое количество тонкого изолированного провода (лучше всего марки МГТФ-0.07).

Дополнительные ИМС установите на свободное место на "слепыше" и подведите к ним питание (+5в) и "землю". Теперь приступим к собственно доработкам:

1) Замените нагрузочные резисторы джойстика на МЛТ-0.125,1кОм. Их общую точку отсоедините от "+5в" и подключите к "земле". С "землей" следует также соединить выводы 6, 10 и 13 мультиплексора D37 (D42) К555КП11. Общий провод джойстика при этом соединяется с "+5в". Это необходимо для использования джойстиков, изготовленных в стандарте "Atari", которые имеют нормально разомкнутые контакты ("Ленинград-1" спроектирован под самодельный джойстик с нормально замкнутыми контактами). Разумеется, что если Ваш джойстик уже подключен через инверторы, то резисторы менять не обязательно, а свободные входы мультиплексора заземлить все равно придется.

Примечание: автору известны 2 варианта чертежа принципиальной схемы, которые отличаются только нумерацией корпусов ИМС, поэтому приводится двойная нумерация. Вам следует определить, какая из них имеется у Вас. Для справки: упомянутый мультиплексор установлен на плате рядом с разъемом кабеля клавиатуры, ближе к краю платы.

2) Обеспечим стандартную адресацию порта вывода 254, выполненного на ИМС D39 (D40) К555ТМ9. Для этого произведем следующие изменения в схеме компьютера (рис. 1):

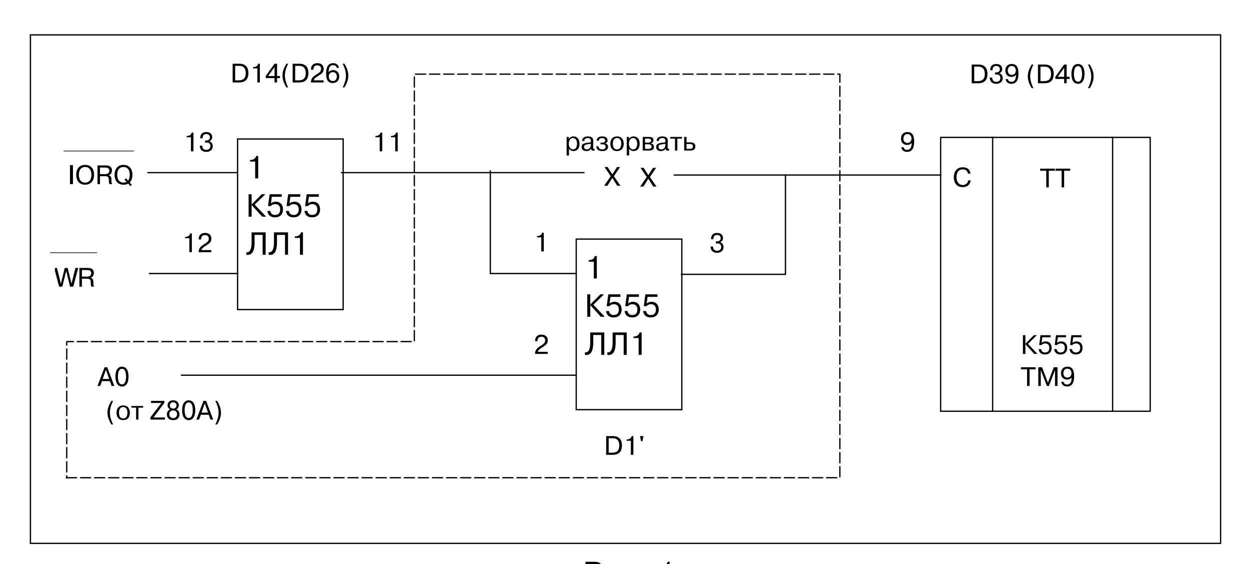


Рис. 1

Этим Вы блокируете запись в регистр К555ТМ9 при обращении по "нечетному" адресу, здесь и далее дополнительно введенные элементы имеют после номера значок " ' " чтобы отличать их от элементов базовой схемы компьютера.

3) Обеспечим нормальную адресацию интерфейса Kempston-джойстика. Для этого служит следующая доработка (рис. 2):

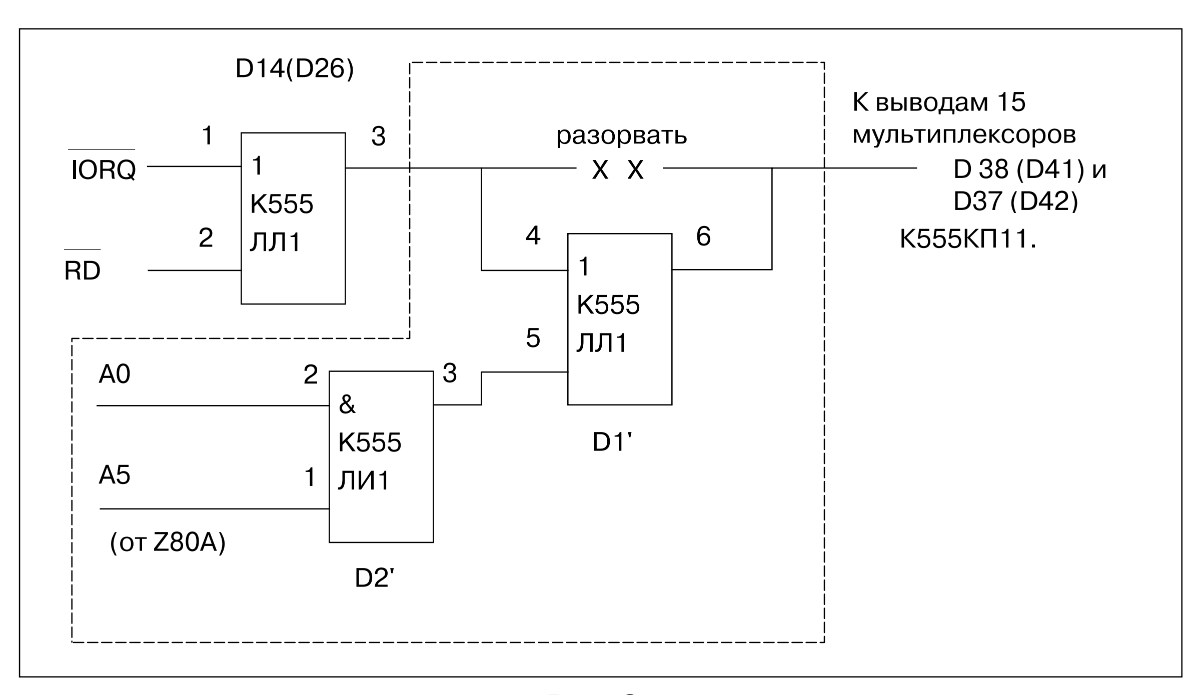


Рис. 2.

При этом обеспечивается "фирменная" выборка по следующему принципу:

- при A0="0" - клавиатура;
- при A0="1" и A5="0" - джойстик;
- при A0="1" и A5="1" - ничего не выбрано, при этом с шины данных считывается байт 255 (#FF).

Вышеприведенные доработки не обеспечивают 100% совместимости с фирменным ZX-Spectrum (следует отметить, что абсолютно совместимых моделей самодельных Спектрумов на сегодня не существует, - причины этого достаточно сложны и их обсуждение не является задачей данной заметки), но позволяет ликвидировать подавляющее большинство конфликтных ситуаций между аппаратной частью и программным обеспечением компьютера.

Еще несколько примечаний. Во-первых, помните, что "Ленинград- 1" - достаточно сложная машина, которую нетрудно вывести из строя неумелым вмешательством в схему и нелегко потом отремонтировать, поэтому не беритесь за доработку, если Вы не обладаете достаточным опытом работы с микропроцессорными системами, лучше обратитесь за помощью к более опытному человеку. Во-вторых, если Вас не волнуют проблемы с джойстиком, можете выполнить только пункт 2 (изменение выборки порта вывода 254). В-третьих, убедитесь перед началом доработки, что монтаж совпадает со схемой, и ранее до Вас никакого вмешательства в схему компьютера произведено не было.

Доработка "Pentagon-48K" для обеспечения совместимости с Sinclair "ZX-Spectrum"

На Spectrum в версии "Pentagon-48K", собранной в соответствии со схемой, не работают некоторые программы, например, "ELITE", копировщики COPY-86M, OutCopy. Одна из вероятных причин этого, возможно, заключается в следующем.

Известно, что микропроцессор Z80A имеет 3 режима обработки прерываний: IM0, IM1 и IM2. С обработкой прерываний в первых двух режимах проблемы не возникает, поскольку и в том, и в другом случае производится переход по фиксированному адресу памяти, начиная с которого размещен обработчик прерывания. При обработке прерывания в режиме IM2 периферийное устройство, выдавшее сигнал INT, должно установить на шине данных младший байт адреса, по которому в памяти находится двухбайтный вектор адреса перехода на обработчик прерываний этого устройства. Этот байт считывается процессором. Старший адресный байт, используемый для формирования полного адреса обработчика прерываний, берется процессором из регистра I. Таким образом, в памяти может быть организована таблица векторов (точек входа в подпрограммы-драйверы

внешних устройств), а сами драйверы располагаются произвольно, без привязки к жестко заданным аппаратной логикой адресам.

Хотя в компьютере ZX-Spectrum и не предусматривался режим IM2, он вполне может быть использован при соблюдении определенных ограничений. Это основывается на том, что в случае отсутствия на шине данных по заданному адресу устройства, которое эти данные выставляет, с нее считывается байт #FF (255), поскольку шина в фирменном компьютере "подтянута" резисторами 8.2 кОм к "плюсу" источника питания. Единственным стандартным источником прерываний в Spectrum является синхронизатор дисплея, выдающий 50 запросов на прерывание в секунду (в начале развертки каждого телевизионного кадра). Естественно, никаких данных на шину он при этом не выставляет, поэтому процессор примет #FF за младший байт адреса, по которому размещен вектор перехода на обработчик прерывания.

Таким образом, мы имеем возможность "перехватить" управление у стандартного драйвера прерывания, записанного в ПЗУ машины. Для этого необходимо выполнить следующее:

- разместить свой драйвер обработки прерывания в произвольно выбранном Вами месте памяти;

- вектор перехода, по которому производится передача управления драйверу, записать в ОЗУ, начиная с адреса #xxFF (т.е. старший байт адреса произволен, а младший всегда равен 255);

- записать старший байт (#xx) в регистр I процессора;

- выполнить команду IM2. Теперь при каждом прерывании будет запускаться не подпрограмма в ПЗУ, а Ваш собственный драйвер.

Такое решение может быть полезным, например, если Вы пишете программу, в которой необходима постоянная индикация текущего времени на экране (типа "электронных часов"), но параллельно с этим нужно выполнять и какие-то другие действия (скажем, редактировать текстовый документ), которые должны занимать основную часть процессорного времени. Возможно применение такого способа и при организации нестандартного драйвера клавиатуры и для других "фоновых" задач. Для любителей "покопаться" в коде фирменных программ не секрет, что немало игр могут использовать этот режим прерываний.

В "Pentagon-48K" шина данных не имеет нагрузочных резисторов, в результате чего при чтении байта с несуществующего устройства в некоторые моменты времени ее состояние может оказаться неопределенным. В режиме IM2 это может привести к "захвату" ложного адреса обработки прерывания, последствия чего очевидны. Не исключено, что в игре "ELITE" именно это и происходит.

Борьба с этим естественна - ввести в схему 8 резисторов, "подтягивающих" выводы D0...D8 процессора к шине "+5 вольт". Номинал этих резисторов для различных экземпляров машины может быть в пределах 3,3...10 кОм. Разумеется, не следует без достаточных оснований стараться установить резисторы "поменьше", поскольку это увеличивает коэффициент нагрузки и без того перегруженного процессора. Попробуйте сперва 7,5...10 кОм, и только если это не дает эффекта, уменьшайте сопротивление.

Для тех, кто имеет "Pentagon-48K" без собранного контроллера "Beta disk Interface", или с ним, но собранным по стандартной схеме (на плате установлены 4 ПЗУ К573РФ4, РФ6, 2764 или 2 ПЗУ 27128 и шинный формирователь КР580ВА86), переделки на этом заканчиваются.

Тем же, кто использует в качестве ПЗУ микросхемы 27256, заменив микросхему КР580ВА86 на перемычки, придется ее все же установить, иначе от перегрузки то и дело будет выходить из строя К561ЛН1, установленная в контроллере. Но, впаивая КР580ВА86, отсоедините ее вывод 9 (CS) от общего провода схемы и подайте на него сигнал IORQ от процессора Z80A. Это устранит конфликт на шине данных между 27256 и ВА86.

Вышеописанные доработки неоднократно проверены на практике и значительно улучшают совместимость "Pentagon-48K".

*   *   *

На необходимость доработки "Ленинграда" по дешифрации внешних портов указывают и другие авторы. Мы не будем здесь повторять все присланные рекомендации, но приведем несколько полезных советов от Иванова Н.Ю. из Якутии, касающихся других вопросов.

1. RGB-выход на "Ленинграде". (Рис. 3).

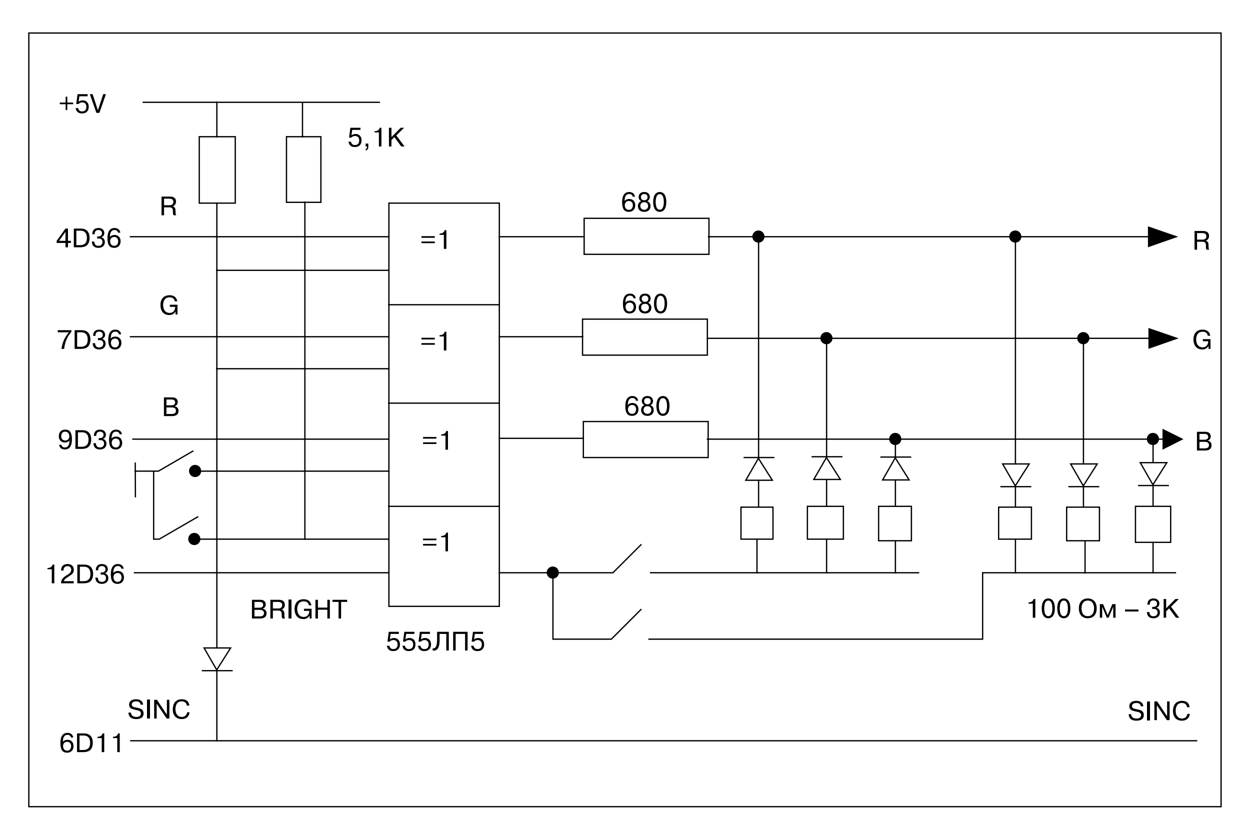


Рис. 3

Микросхема D11 (555ЛП5) желательно должна быть серии 555. Серии же 155 и 1533 работают не вполне хорошо, D40 тоже желательно серии 555.

2. Если на ножку 6 процессора подать 5V через резистор 1 кОм (обязательно по паспорту), то плата будет работать при пониженном напряжении от 4,5 до 5.5 вольт. Были даже случаи и от 3,8 до 4 В.

3. Сигнал INT можно настраивать по музыке из программ Savage или Saboteur. Эти программы очень чувствительны к длительности импульса, а недостаток этот присущ многим заводским машинам.

Проблемы "ELITE".

Мы уже писали о том, что есть возможность перехвата "Таргонов" в гиперпространстве. Многие пилоты пользуются этим для быстрого повышения игрового рейтинга. К сожалению, при этом могут возникать проблемы с нехваткой топлива на Вашем корабле.

Оригинальное решение проблемы нашли пилоты из Екатеринбурга Дремин А. (DEADLY), Кулик С. (DEADLY) и Киселев Д. А. (ELITE) из Казани. Вот их рекомендации:

Перед боем надо заправиться топливом у звезды и оттуда сделать гиперпереход. Следите, чтобы в этот момент надпись FUEL SCOOPS ON была на экране.

Эффект следующий. Во время боев в гиперпространстве все время будет гореть эта надпись, как будто дозаправка топливом не прекращалась. Вы сможете переходить от одной атаки к другой, не боясь "застрять" в космосе. Кстати, это мероприятия одновременно предохранит Вас от утечек горючего при повреждениях в бою (FUEL LEAK!).

*   *   *

Серьезному исследованию подвергли наши читатели поведение недокументированных космических объектов, впервые о которых сообщил нам Кислов Д.Н. из г. Челябинска (ZX-РЕВЮ-92, с.253). Напомним, что это крупные объекты неправильной геометрической формы, выпускающие в случае нападения на них несколько малых боевых аппаратов.

Семья потомственных художников Париловых из знаменитого города Палеха исследовала эти объекты вблизи и пришли к выводу, что они хоть и имеют не вполне регулярную форму, но зато этих форм три. Более того, они различаются даже в месторасположении боевого лазера (конечно это можно установить только побывав в ближнем бою с кораблем, почти вплотную).

Абсолютное большинство писем, которые пришли от пилотов, содержат мнение о том, что это астероид с "отшельниками". Наиболее глубокое исследование подготовил Шилин Д. П. из г. Симбирска. Он подтверждает данные Кислова и значительно их дополняет. Не претендуя на истину в последней инстанции, он систематизировал и обобщил данные и пришел к выводу, что "объект" обладает и свойствами астероида с поселившимися на нем "отшельниками" и свойствами "космической платформы", но факты, говорящие за первую версию, значительно более весомы. Д. П. Шилин высказал еще предположение, что авторы программы включили в нее корабль, обладающий и теми и другими свойствами для того, чтобы в условиях ограниченной памяти дать максимальный простор для фантазии играющих. Поскольку это не первый "фокус" фирмы, то в этой гипотезе что-то есть. Вспомните хотя бы о том, что три дополнительные миссии и связанное с ними оружие были недокументированны в официальной инструкции. Также были недокументированны и ряд управляющих клавиш. Совершенно ясно, что многое фирма "спрятала" для исследователей. Ход очень оригинальный, позволяющий поддерживать интерес к программе годами.

Наш корреспондент предлагает "хаккерам" включиться в исследование программы не только "снаружи", но и как бы "изнутри". Так, в качестве объекта первого удара им предлагается код программы где-то начиная с адреса 58500 и до конца ОЗУ. Здесь находятся текстовые сообщения, в которых, возможно, кто-то и найдет что-либо интересное.

Сообщения закодированы очень распространенным способом, который применяется во многих программах, особенно в адвентюрных.

Известно, что в английском языке многие слова состоят из типичных слогов, например таких, как OR, IR, EA и т.п. Это позволяет уменьшать размеры текстового блока от 1,5 до 5 раз. В ELITE закодированы даже целые слова, повторяющиеся много раз. К примеру, LASER, COMPUTER и другие. Каждому такому слову или слогу присваивается код. Например, присвоим слогу ER код 145, тогда в памяти к примеру вместо слова GLIDER можно будет разместить G, L, I, D, 145. Обычно кодируют 256 слов или слогов. Число 256 принято потому, что в этом случае код будет занимать 1 байт.

Просмотр таких закодированных сообщений может быть трудным. Это не гладкий текст, а обрывки каких-то фраз, череда разных букв. Поэтому этот способ не только сжимает текст, но еще и делает его нечитаемым. Д. П. Шилин исследовал текстовый блок "Элиты" и даже составил программу по его декодированию, но работу еще не закончил в связи с нехваткой времени. Если кто-то найдет там что-либо интересное, мы с удовольствием напечатаем. Сам же корреспондент приводит значения некоторых вскрытых им кодов:

128 - AL	129 - LE
131 - GL	133 - CE
134 - BI	137 - ES
138 - AR	139 - HA
140 - IN	141 - DI
142 - RE	145 - AT
144 - ER	146 - EB
148 - RA	149 - LA
150 - VE	151 - TI

152 - ED	153 - OR
154 - QU	155 - AH
156 - TE	158 - RI
159 - ON	165 - SYSTEM
174 - UNIT	187 - LASER
203 - ST	206 - CARGO
215 - COMPUTER	227 - LARGE

Следует также добавить, что обычно в программах такие коды начинаются с какого-то определенного числа, ведь надо же оставить еще место для таблицы символов. В стандартном знакогенераторе символы расположены, начиная с 32 по 127.

Очень часто в программах для увеличения скрытности текста этот отрезок смещается в какую-либо сторону, то есть коды символов специально делают несовпадающими со стандартными, однако, к счастью, создатели "Элиты" этого не сделали.

Примечание "ИНФОРКОМа":

За примерами того, как кодируется текст в игровых программах далеко ходить не надо. Этот прием чрезвычайно широко распространен. Но ведь кодируют не только текст, но и графику. Мы как раз подготовили статью о том, как это делают на примере программы JET SET WILLY. В одном из ближайших выпусков напечатаем и Вы сможете очень элегантно хранить графику своих программ с малым расходом памяти.

47-ая галактика: тупик или дорога к новый мирам?

Мы неоднократно упоминали о появлении 47-ой галактики в версии Родионова. Большинство наших читателей полагают, что это дефект программы, вызванный неаккуратным снятием защиты. На это указывают и неадекватные суммы кредов на счетах и сумасшедшее вооружение. Но если внешний дефект смог вызвать появление 47-ой галактики, то возникает вопрос: "А нельзя ли нормальным путем проникнуть в новые миры?". По крайней мере в двух письмах начато исследование этого вопроса. Но начнем все по порядку.

Несколько месяцев назад мы предложили читателям начать атаку на тот отгрузочный блок, который сохраняется на ленте при сохранении состояния программы. Многие уже в этом преуспели и мы об этом тоже писали. К чему же ведет это исследование дальше? Давайте посмотрим.

Сначала мы представим выводы, сделанные Балисом Линасом из Каунаса.

1...11 - имя пилота в символах ASCII. Байт, равный нулю информирует об окончании имени.

12 - правовой статус.

0 - clean

1...49 - offender;

50...255 - fugitive.

Как видите, правовой статус - величина не дискретная, т. е. может постепенно ухудшаться или улучшаться.

13...15 - боевой рейтинг. Значения этой трехбайтной переменной таковы:

0,0,0...255,8,0	- HARMLESS
0,9,0...255,16,0	- M. HARMLESS
0,17,0..255,32,0	- POOR
0,33,0..255,64,0	- AVERAGE
0,65,0..255,128,0	- AB. AVERAGE
0,129,0.255,255,2	- COMPETENT
0,0,3...255,255,10	- DANGEROUS
0,0,11..255,255,25	- DEADLY
0,0,26..255,255,255	-ELITE

16...17 - ?

18 - номер карты галактики (0 -первая, 1 - вторая и т. д.).

19 - CASH
1...255 - 1000 - 255000 Cr.
20 - CASH
1...255 - 256000 - 65280000Cr
21 - CASH
1...255 - 0,1 - 25,5 Cr
22 - CASH
1...255 - 25,6 - 6528 Cr
Сумма денег образуется суммированием всех этих четырех байтов.
24...40 - наличие груза в корабле (см. ZX-РЕВЮ-92, с.254).
41 - грузоподъемность корабля. Если здесь содержится 0 и в байтах 24...40 тоже нули, то в корабле находятся беженцы (REFUGEES).
42 - FRONT LASER:
1 - pulse laser:
2 - beam laser;
3 - military laser;
4 - mining laser.
43 - REAR LASER
44 - LEFT LASER
45 - RIGHT LASER
46 - байт первой миссии. Если не 0, то дают миссию.
47 - FUEL
48 - количество ракет.
49 - LARGE CARGO BAY (В номере ZX-РЕВЮ-92, с. 254 здесь и далее ошибка).
50 - E. C. M. SYSTEM
51 - Дополнительный PULSE LASER
52 - Дополнительный BEAM LASER
53 - FUEL SCOOPS
54 - ESCAPE POD
55 - ENERGY BOMB
56 - ENERGY UNIT
57 - DOCKING COMPUTER
58 - GALACTIC HYPERDRIVE
59 - Дополнительный MILITARY LAS.
60 - Дополнительный MINING LASER
61...66 - данные по галактике, в которой Вы находитесь.
67 - ?
68 - Координата Y курсора на большой карте галактики.
0 - вверху; 255-внизу.
69 - ?
70 - Координата X курсора на большой карте галактики, 0 - слева; 255 - справа.
71 - 74 - ???
75 - 91 - содержат информацию о наличии товаров на станции.
92 - ?
93 - наличие дополнительного оружия на корабле:
64...127 - CLOAKING DEVICE;
128..191 - E.C.M. SYS. JAMMER;
192..255 - и то и другое.
94...102 - ??

Теперь рассмотрим внимательно байты 61... 66, содержащие данные по галактике, в которой Вы находитесь.

	1	2	3	4	5	6	7	8
61	74	148	41	82	164	73	146	37
62	90	180	105	210	165	75	150	45
63	73	144	33	66	132	9	18	36
64	2	4	8	15	32	64	128	1
65	83	166	77	154	53	106	212	169
66	183	111	222	189	123	246	237	219

Обратите внимание на то, что содержимое данных по каждой галактике можно получить логическим удвоением данных по предыдущей галактике. Мы говорим именно о логическом, а не об арифметическом удвоении именно потому, что если при удвоении образуется число, большее чем 255, то 255 от него отбрасывается, а в таблицу заносится остаток.

	41	42	43	44	45	46	47	48
61	129	3	6	12	24	48	96	192
62	161	67	134	13	26	52	104	208
63	245	235	215	175	95	190	125	250
64	72	144	33	66	132	9	18	36
65	80	160	65	130	5	10	20	40
66	241	227	199	143	31	62	124	248

Таким образом, облетев все восемь галактик и полетев дальше, Вы попадаете в ... первую галактику.

И вот что по этому поводу пишет наш читатель из Каунаса:

"Если изменить значение хотя бы одного байта, мы попадем в новую систему из восьми галактик и названия планет не будут повторяться. Возможно, что в игре есть какие-то условия, позволяющие перелететь в другие системы, но я их пока не обнаружил. Сколько всего галактик в игре, я даже не берусь ответить. Прошу всех о помощи! Надо найти способ перелета в другие галактики! Я уверен, что он есть, и большая вероятность, что там скрывается планета RAXXLA. "

Мы не случайно приводим слова нашего читателя из Литвы дословно. У Вас есть прекрасная возможность сравнить его выводы с выводами, полученными в то же время, но на несколько тысяч километров восточнее.

Семейный экипаж из села Гайтер Хабаровского края в составе Троеглазова Павла Герасимовича и его сына Геры провел огромную многодневную работу. Облетев восемь галактик, не уклоняясь ни от одного боя и отгружаясь на каждой планете они тщательно исследовали 102-байтный блок и тоже раскрыли циклический характер изменения байтов с 61 по 66 при переходе от 1-ой до 8-ой галактик. Но они пошли дальше и исследовали галактику N47, в которую на некоторых версиях "Элиты" можно попасть "нелегальным" путем. Как оказалось, с ней тоже связана группа из восьми галактик, данные по которым представлены ниже:

И вот, что пишут они:

"А где галактики между 8-ой и 41-ой? Они есть! Мы обнаружили за короткое время еще 36 галактик и прогулялись по ним. Мы также подозреваем, что галактик может оказаться больше, чем 48... В глубинах этих неизвестных еще галактик можно, по-видимому, повстречать и корабли поколений и пресловутую планету RAXXLA. Ищите и обрящите"

Вот как разыскивались эти новые галактики. В первоначальном варианте (система LAVE) исследователи обнуляли один за другим каждый байт из шести (61...66) и каждый раз попадали в новую галактику. Пока еще неясно является ли каждая из вновь открытых галактик "опорной" для генерации целой серии из 8-ми галактик или эти галактики - часть одной системы. Не выяснены также все вопросы, связанные с "устойчивостью" новых систем. Дело в том, что после перехода из первой во вторую и т. д. и вновь после возврата в первую могут наблюдаться сбои в работе программы. Возможно, есть какие-то законы, ограничивающие бесконечно возможное количество галактических систем, зато какие

открытия там возможны! Так, например, при 61-ом байте, равном нулю, в самом центре 5-ой галактики есть ЧЕТВЕРНАЯ ЗВЕЗДА:

RAGETI

ADIIS TEAMA

BESOORRA

Обнулив 62-ой байт и прогулявшись по полученной "восьмерке", наши корреспонденты с интересом увидели, что во второй галактике созвездия неожиданно похожи на земные: М. Медведица, Цефей, Дракон, Рыба, Лира. Третья галактика - необычно высоко социально развита, в шестой есть созвездие, похожее на Северную корону, а восьмая - кишит бандитами всех мастей.

Вот к таким интересным открытиям может привести кропотливое исследование. Теперь дело совсем за малым - надо найти способ и условия возможности перемещения между галактическими системами. Но этот малый шаг можно пройти только объединив усилия всех пилотов и растянуться он может надолго.

Авторы исследования полагают, что они затронули только "краешек" глобальной проблемы доведения игры до логического конца.

"Новые" галактики - это еще не все интересное из того, что удалось установить семейному экипажу. Интерпретация байтов в основном та же, что и представленная выше, но зато есть некоторые любопытные комментарии.

1. Байты 24 - 40 - товары.

Здесь есть особенность. Максимальные цифры (255) можно вносить только в 37-ой, 38-ой и 39-ый байты. Сумма же остальных байтов (товаров) не должна превышать 20 (35, если у Вас есть LARGE CARGO BAY). Все же, что выше этой нормы, будет при попытке продажи умножено на существующую в данном месте цену и списано с Вашего банковского счета. А вот, что будет, если Вы доведете свой счет таким способом до нуля и будете продавать дальше, - узнайте сами.

2. Байты 42... 44. Засылая сюда число, большее чем 4, Вы получаете лазеры с самыми невероятными названиями. 99% их работают, как технологические и все имеют хорошую скорострельность и огромную силу удара. Замечено, что бортовые лазеры имеют примерно вдвое более высокую скорострельность, чем носовые и кормовые.

Интересно, что на этих "супер-лазерах" можно хорошо подзаработать. При замене такого оружия на стандартное (из меню) на Ваш счет в банке поступает немалая сумма (30 тыс. и более) компенсации, но бывает и наоборот. Проверьте!

3. Байт 46 - миссия "Сверхновая".

Обычное значение - 0. В зависимости от величины засланного сюда числа, Вы получите миссию сразу или после нескольких перелетов с планеты на планету.

4. Интересно поэкспериментировать с байтами из группы 47-60.

Засылая в 47-ой байт число 255, Вы получите на карте круг размером 25,5 световых лет и сможете перелететь на самую дальнюю планету этого круга за один обычный перелет, но если перелетите на ближайшую, излишки топлива у Вас отберут и останется обычный радиус в 7 световых лет.

Заслав в 48-ой байт число 100 или 200, Вы получите соответственно 100 или 200 ракет. При пуске ракет они могут появляться стаей на экране, заслоняя "пейзаж". Можно стрелять очередями по 5-6 ракет.

Во все байты, кроме 50-го можно засылать максимальное число -255.

5. С группой байтов 94...102, увы и этому экипажу справиться пока не удалось. Что ж, будем ждать новых открытий.

Кто может помочь с ремонтом японского дисковода TEAC - отзовитесь. Вышла из строя 40-ножечная микросхема на плате 15532097-05B.

745100, Туркмения, Балканская обл., г. Небит-Даг, кв-л 211, д. 70, кв. 14, Виннику.

СОВЕТЫ ЭКСПЕРТОВ

Дорогие друзья!

Много читателей с интересом следят, как развивается "раскрутка" программы ELITE на страницах ZX-РЕВЮ. Вместе с тем, они справедливо отмечают, что давно бы пора найти этой "программе десятилетия" достойную замену и дружно предлагают на эту роль известную программу "ACADEMY" (развитие программы TAU CETI).

Сегодня мы предлагаем Вашему вниманию необычную экспертную проработку, выполненную сразу двумя авторами. Они проживают в разных городах и, возможно, даже не знают друг друга, но их объединяет любовь к этой программе. Мы же взяли на себя скромную задачу компиляции представленных ими материалов в единый блок.

Фамилии экспертов приведены в порядке поступления материала.

ACADEMY (TAU CETI II)

Автор: Pete Cooke.
Фирма: CRL Group PCL.
Год: 1986.

Эксперты:
Хоминич Р.В., г. Киев
Жаров Р.Н., г. Херсон



Академия Галактической Корпорации по повышению квалификации пилотов скиммеров (GASP) была основана в 2213 году после несчастного случая на 61 Cygnus, когда пилот новичок, выбрав неисправный аппарат, ошибочно произвел стыковку с реактором и половина планеты погибла под расплавленной лавой. Галактическая Корпорация приняла решение создать специальный тренировочный центр подготовки элитного корпуса пилотов, летающих на новейших военных скиммерах для применения в колониях и аванпостах Вселенной.

Выпускники академии направлялись в ряды специального корпуса скиммеров, где кандидатам требовалось успешно выполнить двадцать заданий, состоящих из пяти уровней, по четыре задания каждый.

Будьте внимательны! И, возможно, Вас будет ждать награда Галактической Корпорации.

Итак, прочитав сценарий игры, Вы нажимаете "FIRE" и переходите в главное меню.

Работа с меню.

Выбор осуществляется путем перемещения стрелки курсора и нажатием "FIRE". Выбранная Вами опция окрашивается в белый цвет.



Главное меню имеет следующие опции:
Accept Mission - выполнение миссии;
Select Mission - выбор миссии;
Select Skimmer - выбор скиммера;
Progress Report - рапорт о выполнении уровня;
Tape Menu - меню работы с лентой;
Enter a New Cadet - прием нового кадета;
View/Redefine Keys - просмотр/выбор клавиш;
Кроме этого в меню указаны: 00:00:00 - часы;
Sound - звук: "V"-вкл. , "x"-выкл;
Mission - текущая миссия;
Skimmer - выбранный скиммер;
Cadet - имя и фамилия кадета.

Просмотр/выбор клавиш.

Клавиши управления:
O - влево; P - вправо;
S - вверх/ускорение;
X - вниз/торможение;
N (SPACE) - огонь лазера/выбор;
M - огонь ракетой;
A - огонь противоракетным снарядом;
F - огонь осветительной ракетой;
B - сбрасывать бомбу;
V - круговой осмотр;
H - увеличение высоты;
G - уменьшение высоты;
J - прыжок;
L - приземление;
I - инфракрасное видение;
R - доклад о состоянии систем корабля.
Alter Keys - выбор клавиш: Kempston Joystick - джойстик;
Return To Main Menu - возврат в главное меню;
"BREAK" - восстановление стандартных клавиш.

Если у Вас подключен джойстик, то он будет выбран автоматически. Не рекомендуется менять клавиши управления (кроме первых пяти) во избежание путаницы.

Прием нового кадета.

Прошение о приеме в группу подготовки пилотов скиммеров.
Форма: VV1B/6702 (3 экземпляра).
Дата прошения 7/11/2047.
Имя (имя и фамилия): ....
Дата рождения (день/месяц/год): ../../....

Начать с I уровня (да/нет): ...

Для заполнения бланка введите свои имя и фамилию, а затем дату рождения. Год, видимо, надо рассчитывать, учитывая сегодняшнюю дату: 2047 год. Поэтому, если Вам 20 лет, то Вы родились в 2027 году и т. д. Вы можете выбрать любой год с 1901 по 2040. После введения даты в скобках появится день недели Вашего рождения.

Последний вопрос имеет смысл, если Вы находитесь на II-ом уровне или выше.

Меню работы с лентой.

Load Game File - загрузить отложенную ситуацию.
Save Game File - записать текущую ситуацию.
Load Ship Designs - загрузить созданный корабль.
Save Ship Designs - сохранить созданный корабль.
Return To Main Menu - возврат в главное меню.

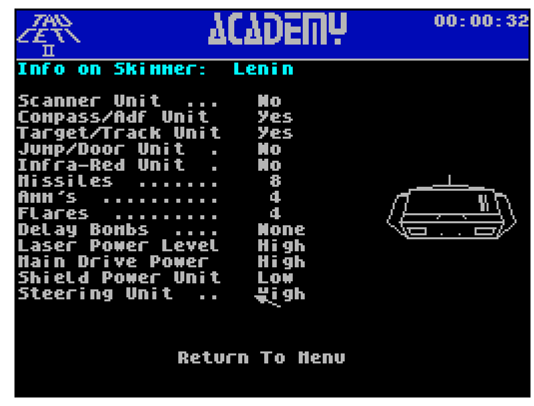
Рапорт о выполнении уровня.

В рапорте указаны имя кадета, уровень игры и степень выполнения каждой миссии.

Миссии считается выполненной, если Вы набрали не менее 90%. Уровень считается пройденный, если Вы набрали средний результат, близкий к 100%, выполнив все четыре задания.

Выбор скиммера.

Вы можете выбрать один из трех стандартных скиммеров:
GCS Lenin (Ленин),
GCS Lincoln (Линкольн),
GCS Wilson (Вильсон).



Кроме этого, Вы можете спроектировать скиммер по своему усмотрению (одновременно в памяти может быть три личных скиммера).

Тактико-технические характеристики стандартных скиммеров Вы можете просмотреть с помощью функций приведенных ниже:

Info on this Skimmer - информация о скиммере, отмеченном "V";
View Panel - просмотр панели скиммера;
Design New Skimmer - спроектировать новый скиммер;
Selection Complete - выбор завершен.

Проектирование скиммеров.

Для своего скиммера Вы можете выбрать любое оборудование, соблюдая два условия: его стоимость не должна превышать 100 MCr и его вес не должен превышать 100 единиц.

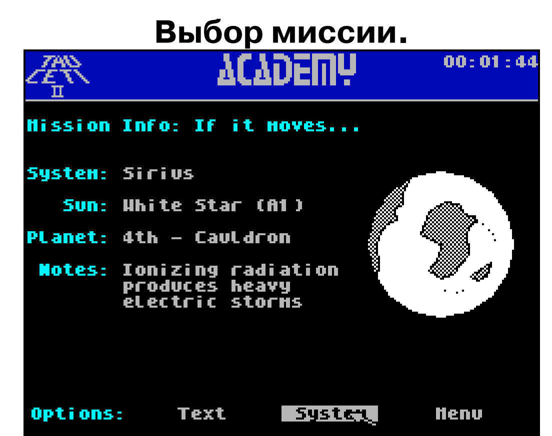
После выбора Design New Skimmer, перед Вами появится таблица:
Scanner Unit - сканнер;
Compass/Adf - компас/азимут;
Target/Track Unit - цель/курс;
Jump/Door Unit - гиперпрыжок/стыковка;
Infra-Red Unit - инфракрасное видение;

Missiles - ракеты;
Amm's - противоракетные снаряды;
Flares - осветительные ракеты;
Delay Bombs - бомбы замедленного действия;
Laser Power Level - уровень мощности лазера;
Main Drive Power - главный энергетический привод;
Shield Power Level - энергия защиты;
Steering Unit - блок управления.

Отметьте знаком "v" нужный ответ:
No (нет); Yes (да);
None (не требуется); 4 (штуки);
8 (штук); Low (низкий);
Med (средний); High (высокий).
После чего выберите Design Complete (проектирование завершено).
Если Вы решили отказаться от проекта, то выберите Abandon Design (отказ от проектирования).

Вы перешли в режим проектирования панели. Выберите структуру панели и ее цвет, затем расположите приборы (Place Instruments).
Появятся сообщения: Put Viewscreen (разместить обзорный экран) и Undo Last Step (отменить последнее действие).
Выберите место для экрана и поместите туда белый прямоугольник, нажав "FIRE". После чего выберите Menu (меню) и повторите операцию с окном сообщений (Message window). Желтый прямоугольник - поле занято, белый - поле свободно.
Далее размещается выбранное Вами оборудование. Последними размещаются: Height Gauge (шкала высоты), Shield Gauge (уровень защитной энергии), Fuel Gauge (шкала запаса топлива), Laser Temp (температура лазера) и Speed Gauge (шкала скорости).
На этом проектирование завершено. Осталось только присвоить имя новому скиммеру. При желании, Вы можете записать на ленту созданные корабли (см. меню работы с лентой).

Выбор миссии.



Для выбора миссии отметьте ее знаком "v". Используя Info in this Mission (информация о миссии), Вы сможете ознакомиться с целью (Text) и планетной системой (System), в которой Вам предстоит действовать.
Функция Load in Next Level (загрузить следующий уровень) будет заблокирована до полного выполнения всех миссий этого уровня.
Прежде, чем приступить к выполнению выбранной миссии, Вы должны внимательно изучить предоставленную информацию по звездной системе.

Уровень I.

Миссия I: If it moves... (Если оно движется...).
Цель: Уничтожение вторгшихся роботов.

Счет очков: Процентное основание.

Примечание: Нет системы гиперпрыжков, нет системы поддержки, одна (1) G.L.V. база.

Рекомендуемый скиммер: GCS Ленин.

Система: Sirius (Сириус).

Солнце: Белая звезда (A1).

Планета: 4-ая Cauldron (Котел).

Примечание: Ионизирующее излучение тяжелых элементов, электрические бури.

Миссия II: Red Dawn (Красный рассвет).

Цель: Уничтожение автоматических заводов во всех квадратах.

Счет очков: Процентное основание.

Примечание: Полная система гиперпрыжков и центр снабжения, одна (1) G.L.V. база.

Рекомендуемый скиммер: GCS Вильсон.

Система: Betelguese (Бетельгейзе)

Солнце: Красный гигант (M2).

Планета: 6-ая Eventide (Вечер).

Примечание: Гигантское красное солнце, господствующее на небе, делает инфракрасную систему бесполезной.

Миссия III: Meltdown (Плавка).

Цель: Реактор в критическом состоянии - должен быть уничтожен.

Счет очков: 15 минут до расплавления.

Примечание: Нет системы гиперпрыжков, нет системы поддержки, одна (1) G.L.V. база.

Рекомендуемый скиммер: GCS Ленин.

Система: Звезда Ван Маанена.

Солнце: Желтый карлик (dG5).

Планета: Escot.

Примечание: Одноликий мир, маленькая полярная колония.

Миссия IV: Softly Softly (Тихо-тихо).

Цель: Определить местонахождение и возвратится на базу.

Счет очков: По времени.

Примечание: Зона недавно заминирована оборонным сектором Галактической Корпорации. Причина минирования - административная ошибка.

Рекомендуемый скиммер: GCS Линкольн.

Система: Rigel (Ригель).

Солнце: Бело-голубая звезда (B8)

Планета: 12-ая Ice-world (Ледяной мир).

Примечание: Аванпост обороны, нет гражданских сооружений.

Уровень II.

Миссия I: Cipher (Шифр)

Цель: Собрать и смонтировать кодовые устройства реакторов.

Счет очков: Четыре кодовых блока.

Примечание: Нет системы гиперпрыжков, нет системы поддержки, одна (1) G.L.V. база.

Рекомендуемый скиммер: GCS Линкольн.

Система: Vega (Вега).

Солнце: Белая звезда (A0).

Планета: 5-ая Homebase (Родная база)

Примечание: Маленькая, недавно сформированная колония.

Миссия II: At the OK Coral (В загоне все в порядке).
Цель: Уничтожение бродячих роботизированных систем.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков, система поддержки, одна (1) G.L.V. база.
Рекомендуемый скиммер: GCS Ленин.
Система: Avior.
Солнце: Бело-голубая звезда (09)
Планета: 3-ая Corral (Кораль).
Примечание: Одноликий мир.

Миссия III: Where to Guv?
Цель: Пираты захватили гипертранспортную сеть - требуется их уничтожить.
Счет очков: Процентное основание.
Примечание: Полная система гиперпрыжков и система поддержки, одна (1) G.L.V. база.
Рекомендуемый скиммер: GCS Линкольн.
Система: Sirius (Сириус).
Солнце: Белая звезда (A1).
Планета: Greengage (Зеленый заложник).
Примечание: Озоновый слой создает необычные световые эффекты.

Миссия IV: Hide and seek (Прятки).
Цель: Уничтожить комплекс солнечных дисков и вернуться на G.L.V. базу.
Счет очков: Процентное основание.
Примечание: Уничтожение дроидов может быть полезным.
Рекомендуемый скиммер: GCS Вильсон.
Система: Beta Hydri (Бета Гидры).
Солнце: Желтая (G1).
Планета: Engels (Энгельс).
Примечание: Одна из первых 4-х колоний.

Уровень III.

Миссия I: Laserium (Лазериум).
Цель: Уничтожение роботов-захватчиков.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков, небольшая поддержка, одна (1) G.L.V. база.
Система: Groombridge 34.
Солнце: Красная звезда (M2).
Планета: Единственная планета - Rayet (Лучистая).
Примечание: Малая новая колония.

Миссия II: Hades II (Гадес II).
Цель: Обнаружение и уничтожение вышедших из под контроля роботизированных систем.
Счет очков: Процентное основание.
Примечание: Небольшая система гиперпрыжков и центры поддержки, одна (1) G.L.V. база.
Рекомендуемый скиммер: GCS Вильсон.
Система: Procyon (Процион).
Солнце: Белая звезда (F5).
Планета: 15-ая Hades II.
Примечание: Очень удалена от солнца и постоянно покрыта льдом.

Миссия III: The Sands of Time (Песок времени).

Цель: Сеть реакторов должна быть уничтожена до эвакуации планеты.
Счет очков: Временное основание (1 час).
Примечание: Нет системы гиперпрыжков, нет системы поддержки, одна (1) G.L.V. база.
Рекомендуемый скиммер: GCS Ленин.
Система: Fomalhaut (Фомальгаут).
Солнце: Белая звезда (A3).
Планета: Foma-3.
Примечание: Изолированный аванпост.

Миссия IV: Mission Improbable (Миссия "Невероятная").
Цель: Собрать и смонтировать кодовые блоки реакторов и возвратиться на G.L.V. базу.
Счет очков: Процентное основание.
Примечание: Deaf-фильтр применяется во многих реакторах.
Рекомендуемый скиммер: GCS Линкольн.
Система: Delta Pavonis (Дельта Павониса).
Солнце: Желтая (G1).
Планета: 3-я Dawnmist (Утренний туман).
Примечание: без примечаний.

Уровень IV.

Миссия I: Ceti Revisited (опять Тау Кита).
Цель: Сбор и монтаж кодовый блоков реакторов.
Счет очков: По мере сбора блоков.
Примечание: Система гиперпрыжков, небольшая поддержка, одна (1) G.L.V. база.
Система: Тау Кита.
Солнце: Желтая звезда (G8).
Планета: Третья планета.
Примечание: Одна из четырех первых колоний.

Миссия II: Out of the Frying Pan (Из огня ...).
Цель: Обнаружение и уничтожение роботов вторжения.
Счет очков: Процентное основание.
Примечание: Небольшая система гиперпрыжков, полная поддержка, одна (1) G.L.V. база снабжения.
Система: Ригель.
Солнце: Бледно-голубая (O9)
Планета: 9-ая "Безымянная".
Примечание: нет примечаний.

Миссия III: Don't panic. (Без паники).
Цель: Уничтожение реакторов и солнечных дисков.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков, небольшая поддержка, одна (1) G.L.V. база.
Система: Эпсилон Инди.
Солнце: Оранжевая звезда (K5).
Планета: Adams.
Примечание: Радиация делает систему ночного зрения бесполезной.

Миссия IV: Needle in a Haystack (Иголка в стоге сена).
Цель: Определить собственное месторасположение и возвратиться на G.L.V. базу.
Счет очков: По времени.
Примечание: Предбазовая зона усеяна суперминами.

Система: Денебол.
Солнце: Белая (A3).
Планета: "Пыльный шар".
Примечание: без примечаний.

Уровень V.

Миссия I: Coal Mine (Угольные копи).
Цель: уничтожение враждебных роботов.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков и системы поддержки, одна (1) G.L.V. база.
Система: Epsilon Erandi.
Солнце: Красная звезда (K2).
Планета: Третья планета.
Примечание: Миссия на ночной стороне планеты.

Миссия II: PAZ!
Цель: Обнаружение и уничтожение роботов вторжения.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков и системы поддержки, одна (1) G.L.V. база. Роботы могут иметь суперракеты.
Система: Процион.
Солнце: Белая звезда (F5)
Планета: 7-ая "Нью-Марс".
Примечание: нет примечаний.

Миссия III: Protector. (Защитник).
Цель: Уничтожение следящей системы на Дельте.
Счет очков: Процентное основание.
Примечание: Нет системы гиперпрыжков и системы поддержки, одна (1) G.L.V. база снабжения.
Система: Альфа Центавра.
Солнце: Желтая звезда (G2).
Планета: Дельта.
Примечание: Ближайшая колония к Солнечной системе.

Миссия IV: The Shepherd (Пастух).
Цель: Найти и выстроить сторожевые башни рядом с G.L.V. - базой.
Счет очков: Пять сторожевых башен по мере сборки.
Примечание: Все пространство минировано.
Система: Альтаир.
Солнце: Белая (A7).
Планета: "Плато".
Примечание: без примечаний.

Полезные советы по сборке скиммеров.

Стандартные скиммеры не всегда являются удобными, поэтому рекомендуется пользоваться личными кораблями. Для проектирования корабля принимается во внимание наличие системы гиперпрыжков, системы поддержки, класс звезды и, конечно, Ваша цель. Возможно, Вам придется сделать несколько разведывательных вылетов, прежде чем Вы найдете приемлемый вариант.



Ниже приводится описание отдельных систем корабля.

Scanner Unit (радар) - рекомендуется ставить на все корабли, значительно облегчает выполнение задачи. Позволяет обнаруживать объекты и проходы в минных полях. Голубой квадрат в центре - зона непосредственного контакта. Граница квадрата - предельно допустимая дистанция приближения к минным полям.

Compass/Adf (Компас/УНБ Указатель Направления на Базу) - обязательный прибор (хотя в некоторых ситуациях опытный пилот может обойтись и без него). В случае его повреждения вдали от базы, Ваша гибель почти гарантирована. Азимут всегда указывает на Вашу G.L.V. базу.

Targeting/Tracking/Drain: Первый блок - это электронный прицел (может быть заменен или дополнен механическим).

Второй - настройка на радиомаяк.

Третий - указывает на то, что Вы подверглись нападению. Красный цвет индикатора указывает на то, что прибор включен. Последний блок может стать зеленого цвета (близкая опасность).

Jump/Door Unit - блок необходим при наличии системы гиперпрыжков или необходимости стыковки с другими объектами.

Система JUMP активизируется только в непосредственной близости от Jump-площадок. Запускается командой JUMP, красный цвет индикатора Jump указывает, что Вы вошли в зону действия платформы для гиперпрыжков.

Система DOOR нужна для открывания проходов в силовых полях.

Работает автоматически. Для открывания базы она не требуется. Красный цвет индикатора Door указывает, что стыковка запрещена (зеленый - разрешена).

Infra-Red Unit - прибор ночного видения. Дает контуры окружающих объектов. Необходим на планетах с резкой сменой дня и ночи (может быть заменен на осветительные ракеты). Включается клавишей Infra-red.

Missiles - самонаводящиеся ракеты (эффективное оружие). Пуск возможен только при наличии захвата цели. Пуск выполняется клавишей Fire miss.

Amn's - система противоракетной обороны. Энергетический залп возможен после получения сообщения от бортового компьютера о ракетной атаке. Пуск клавишей Fire amm.

Flares - осветительные ракеты. Пуск клавишей Fire Flar. Обеспечивают более хорошую видимость. По сравнению с инфракрасной системой, но имеют ограниченное время действия.

Delay Bombs - бомбы замедленного действия. Это чрезвычайно мощное оружие. Уничтожают все объекты в радиусе действия, в том числе и скиммер. После сбрасывания требуется срочно покинуть заминированный квадрат.

Laser - основное оружие Вашего скиммера. Поражает практически все объекты, но при стрельбе быстро перегревается, поэтому его не следует использовать бесконтрольно.

Main Drive Power – главный энергопривод. Этот блок определяет максимальную скорость Вашего корабля, а также время ее достижения.

Shield Power Unit - отсек энергетической защиты. Это генератор защитного поля. Определяет максимальную интенсивность поля и скорость ее восстановления до номинального уровня.

Steering Unit - блок управления. Определяет инерционность Вашего корабля в управлении.

Выполнение миссии.

Итак, цель определена, скиммер выбран и Вы приступили к выполнению миссии.

Перед вами появилась панель корабля, наверху указано положение скиммера, счет в % и бортовые часы. Компьютер дает Вам сообщение о том, что все системы готовы и просит ввести команду.

Наберите "HELP" для получения подсказки.

Сводка боевых команд

HELP - помощь;
LAUNCH - вылет скиммера;
PAUSE - пауза;
QUIT - выход в главное меню;
SIGHTS ON - прицелы включены;
SIGHTS OFF - прицелы выключены;
WAIT - ожидание (около 5 минут);
EQUIP - снаряжение скиммера;
STATUS - состояние систем корабля (аналог "R");
CODES - коды запирающей системы реактора;
DEAF - клавишный электронный звуковой фильтр;
LOOK – осмотр.

EQUIP - в этом режиме Вы можете пополнить запасы оружия и произвести дозаправку (Refuel) и ремонт (Repair). Режим работает только в состоянии стыковки с базой или ремонтным центром.

STATUS - информация о состоянии бортового оборудования.

LOOK - информация о местонахождении корабля.

DEAF - включение Цифрового Электронного Аудио Фильтра, потренируйтесь с этим устройством. Оно сделано специально для защиты от роботов, поскольку они страдают глухотой и дальтонизмом. Этот режим включается автоматически при попытке стыковки с объектом. Для нахождения пароля Вам необходимо нажать Play и затем с помощью курсора

повторить звуковую мелодию. В случае неудачи Вы можете запросить другую мелодию с помощью Reset.

CODES - этот режим предназначен для монтажа кодовых блоков к реакторам. Используется в трех миссиях (2-1, 3-4, 4-1). Облетев реактор, Вы получаете определенное количество кодовых фишек. Обойму можно просмотреть в правом окне командами "вверх"/"вниз". После просмотра Вы с помощью команд Undo и Place переносите какой-либо блок в левое окно. Ваша задача совместить три кода для создания устойчивого рисунка. Изменяя положение и цвет (с помощью Flip и Colour) левого блока, Вы пытаетесь совместить его с каким-либо блоком из правого окна. (Этот этап практически является решением головоломки). Конечным результатом должно стать получение рисунка двух цифр, после чего первая часть кода будет зафиксирована в памяти. Следует повторять операцию до полного нахождения всех цифр.

Если все сделано правильно, код займет соответствующее положение, в противном случае компьютер сообщит об ошибке и укажет на ее причину. Появившиеся цифры будут занесены в графу Locking System Code: _ _ _ _ _ _ _.

Миссия считается выполненной, если заполнены все прочерки.

(Кроме перечисленного компьютер реагирует на команду REACTOR, видимо уцелевшую от игры TAU CETI. При наборе команд, иногда допускаются грамматические ошибки).

Набрав команду LAUNCH, Вы оказываетесь на поверхности планеты и можете приступить к выполнению миссии. Во время боя компьютер информирует Вас о ракетной атаке, о применении роботами Amm's против Ваших ракет, об атаке камикадзе, о Ваших повреждениях, а также выдает некоторую другую информацию.

Итак, Ваша задача - получить звание пилота скиммера. Это произойдет, когда на экране Вашего компьютера появится надпись CADET QUALIFIED.

Если Вы более или менее научитесь управлять скиммером и перейдете к выполнению миссий, Вам могут пригодиться следующие полезные советы:

- КАТЕГОРИЧЕСКИ ЗАПРЕЩАЕТСЯ открывать огонь по платформам гиперпрыжков, центрам поддержки и реакторам! Автоматика ремонтных баз и реакторов, если Вы ее повредите, заблокирует проходы в силовом поле. JUMP-платформы легко повреждаются даже огнем лазера, но это сразу ставит под сомнение возможность выполнения миссии.

Практический опыт показывает, что без вреда (и без пользы) можно обстреливать только свои G.L.V. базы - они не повреждаются даже бомбой, но не пробуйте их таранить. База есть база. Стыковаться лучше на скорости не более, чем 1/4 от максимальной и ниже.

- не пытайтесь расстреливать минные поля - не хватит здоровья!

- не летите на полной скорости к неизвестному объекту (тем более к группе)!

- не забывайте своевременно возвращаться на базу для ремонта и заправки!

- при отмеченном большом количестве целей (радаром или визуально) желательно двигаться импульсами. Роботы обычно наступают шеренгами по 3-4 робота в каждой. Активизируйте шеренги по одной. Так Вы легко расправитесь с нападающими, но если Вы сразу активизируете 2-3 шеренги, Вам не удастся даже развернуться, чтобы отважно скрыться от погони.

- в некоторых миссиях при достаточном удалении от базы желательно фиксировать свое направление по компасу, а еще лучше - по Солнцу. Компас и радар - приборы очень хрупкие и в бою нередко ломаются, а без них, если Вы не знаете направления на базу, лучше начинать миссию заново.

- некоторые миссии Вы пройдете с первого раза, над некоторыми Вам придется поломать голову. Не отчаивайтесь и не ждите подсказок, из любой ситуации есть выход. Лежит он на поверхности, надо только его увидеть.

- помните, что при выходе в главное меню Ваши очки теряются!

Желаем Вам удачи!

SHERLOCK

Наши читатели помнят, как в шестом номере ZX-РЕВЮ за прошлый год мы давали полную фирменную инструкцию к программе "SHERLOCK". Программа заинтриговала многих наших читателей, но несмотря на дружные усилия до сих пор пока никому не удалось раскрыть это сложное и запутанное дело.

А дело действительно интересное. К счастью, совсем недавно, вытирая пыль в кабинете Холмса в доме на Бейкер-стрит 221 Б праправнучка миссис Хадсон случайно обнаружила пакет, датированный 1892 годом, на которой было написано "Вскрыть через сто лет в присутствии сэра К.Синклера и ответственного представителя "ИНФОРКОМа".

После вскрытия из конверта выпали пожелтевшие листочки. Давно выцветшие чернила и малоразборчивый почерк не помешали установить, что это заметки, которые велись по ходу расследования Лизерхэдской трагедии. К сожалению, судя по почерку, их писали не доктор Уотсон и не сам великий сыщик, так что никакой литературной обработки они не прошли. Сведения отрывочны, изложены небрежно. Кто этот неизвестный секретарь, посвященный в дела Холмса, еще предстоит разобраться биографам. Мы же спешим их опубликовать в надежде на то, что эти заметки помогут и Вам докопаться до сути самого захватывающего дела Холмса.

Понедельник, 8:00

Холмс и Уотсон в своей гостиной. По просьбе Холмса Уотсон раскрывает газету и находит в ней сообщение об убийстве молодой женщины в Лизерхэде (SAY TO WATSON "READ CHRONICLE"). Холмс сразу же со всей присущей ему энергией начинает готовиться к расследованию дела.

Пройдя в гардеробную (OPEN PLAIN DOOR), он берет с вешалки два комплекта маскировочной одежды (DISGUISE). Первый комплект - для маскировки под китайца, второй - под старика. Чтобы взять костюм с собой, ему приходится сначала его надеть (WEAR), а затем снять (TAKE OFF). Вернувшись в комнату и поговорив с Уотсоном, Холмс направляется на место преступления и просит Уотсона следовать за ним (FOLLOW ME).

Ближайший поезд в Лизерхэд, как выясняется из расписания, отходит со станции Kings Cross, в 9 часов 15 минут. Чтобы успеть на него, Холмс, выйдя на улицу, должен взять извозчика (HAIL A CAB), сесть в экипаж (CLIMB INTO CAB) и объяснить ему, куда надо ехать (SAY TO CABBY "GO TO KINGS CROSS ROAD").

На третьей платформе Холмс встречается с инспектором Лестрейдом, который тоже следует на место преступления. Подождав до 9:15, сыщики садятся в поезд и в 10:30 прибывают в Лизерхэд.

Здесь Холмс предоставляет Лестрейду вести дело так, как ему кажется нужным. Он ходит за ним по городу (FOLLOW LESTRADE) и внимательно слушает все разговоры.

Расследование привело их к небольшому мосту, сделанному из песчаника. Здесь, на этом мосту и произошло преступление. Тело жертвы еще не убрали в ожидании полиции. Как Холмсу и Лестрейду удалось уже установить, это труп миссис Браун. Более внимательный осмотр места происшествия позволяет найти скомканную записку, подписанную инициалами T.F.

Миссис Браун была убита с близкого расстояния, пуля вошла в правый висок. Возможно, миссис Браун держала оружие в руках, поскольку на правой руке есть следы пороховых газов.

Холмс начинает подозревать, что здесь имело место не убийство, а самоубийство, у Лестрейда же определенно складывается иное мнение.

Проследовав за Лестрейдом к дому покойной, Холмс становится свидетелем допросов близких и домашней прислуги. Эти допросы помогли установить следующее:

- миссис Браун была вдовой недавно умершего мистера Брауна;
- мистер Браун был крупным ученым. Последнее время он работал над неким

секретным военным проектом;

- после его смерти чертежи и прочая документация по проекту бесследно исчезли;

Кроме того, Холмс убеждается во время этих допросов, что мистеру Базилю Фиппсу доверять нельзя, несмотря на то, что он имеет твердое алиби на момент смерти миссис Браун (он весь вечер находился дома и играл на рояле этюды Шопена).

Кроме того, Холмс проводит беседу с майором Персивалем Фосом, проживающим на улице Sidmouth Street, который отказывается ему сообщить, где находился во время смерти миссис Браун.

Инспектор Лестрейд возвращается в Лондон. У него уже есть своя версия убийства и он подозревает в преступлении майора.

Обследуя дом, Холмс обнаруживает еще один труп. В библиотеке он находит тело миссис Джонс. Тщательно обследовав книжные шкафы, Холмс также обнаружил потайную комнату. В этой комнате хранилась женская одежда с пятнами крови. Изучение одежды и меток на ней показало, что принадлежит она Тришии Фендер (Tricia Fender).

При внимательном обследовании письменного стола (desk) в кабинете (study) Холмс обнаруживает ящик с двойным дном, здесь находятся банковские счета миссис Браун. Здесь же есть письмо от Тришии Фендер.

Судя по состоянию банковских счетов, миссис Браун в последнее время неоднократно снимала немалые суммы денег.

Теперь Холмс хочет поподробнее узнать о Тришии Фендер. Он возвращается в гостиную и продолжает допросы свидетелей (TELL ME ABOUT TRICIA FENDER).

Ему удается получить следующую информацию:

- Тришия Фендер была секретарем мистера Брауна;
- она проживает в Лондоне на улице Portman Street;
- она и миссис Джонс очень похожи друг на друга внешне.

Далее Холмс занимается проверкой свидетеля Базиля Фиппса. Он приходит к нему в квартиру на Cobden Lane и проходит наверх, в спальню. Здесь ему становится ясно, что ни на каком рояле свидетель не играл в ночь гибели миссис Браун. У него есть граммофон, на котором стоит пластинка с этюдами Шопена.

Из окна свисает простыня. Можно предположить, что кто-то использовал окно для выхода из дома.

Доверять Базилю Фиппсу нельзя, из свидетеля он превращается в подозреваемого.

Теперь Холмс имеет всю необходимую ему информацию и возвращается в Лондон. Его ближайшая задача - допросить отставного майора, но сначала он забегает к Лестрейду в Скотланд-Ярд (на улице Parliament Street) и решительно заявляет ему, что майор невиновен (THE MAJOR IS INNOCENT).

Лестрейд требует доказательств, ведь майор отказался сообщить где он был во время убийства. Вместе они едут к майору домой на Sidmoutn Street.

Хозяина дома не оказалось и сыщикам пришлось ждать до 11 часов вечера, когда появился майор. Не задержавшись дома, он тут же снова вышел и, наняв кэб, приказал екать на Slater Street. Холмс и компания последовали за ним.

Слежка привела их к притону, в котором собираются курильщики опиума. Майор прошел внутрь. Для того, чтобы пройти за ним, Холмс должен переодеться в костюм китайца.

Теперь Холмсу понятно, почему майор скрывал, где он находился во время трагедии на мосту. Выйдя из курильни, Холмс говорит Лестрейду "THE MAJOR IS IN AN OPIUM DEN". Вскоре появляется и сам майор. Лестрейд отпускает его и говорит Холмсу "WELL DONE, HOLMES".

Так закончился первый день следствия.

Вторник.

Холмс возвращается в Лизерхэд и сразу же направляется к Базилю Фиппсу. Находит там комнату, в которой установлен сейф, вскрывает сейф и обнаруживает письма от Тришии Фейдер к миссис Браун. Из содержания писем становится ясным, что здесь имел место

шантаж. По-видимому, Тришия похитила документы ученого и впоследствии шантажировала его вдову. На это же указывает и состояние банковских счетов миссис Браун.

Чтобы встретить Лестрейда, Холмс идет на станцию. Лестрейд прибывает около 9 часов утра. Здесь Холмс сообщает инспектору, что миссис Браун совершила самоубийство под влиянием шантажа: "MISSIS BROWN KILLED HERSELF".

Вместе они следуют на мост, ставший местом ее гибели. Тщательно обследовав ручей (CLOSELY EXAM THE DEAP STREAM), они находят пистолет, к которому привязан тяжелый камень. Очевидно, совершая самоубийство, миссис Браун хотела наказать шантажистку. Для этого она привязала пистолет к камню, чтобы после выстрела оружие исчезло навсегда. Для этого же она зажала в руке скомканную записку, наводящую на след T.F. То есть, она инсценировала убийство самой себя.

Мост сложен из мягкого камня, песчаника, и падая, пистолет оставил выщербину в нем. Именно это и побудило Холмса тщательно обследовать поток.

Теперь Холмс вновь идет на станцию, ему пора возвращаться в Лондон. Лестрейд соглашается, что миссис Браун совершила самоубийство и говорит: "WELL DONE, HOLMES".

Не позже, чем в 11:15 Холмс должен выехать в Лондон и прибыть на вокзал Кингс Кросс в 12:30. Наняв кэб, он едет на улицу Portman Road к дому, где живет Тришия.

Войдя в дом, он проходит в гостиную. У стены стоит сейф. Здесь его встречает хозяйка. Не обращая внимания на ее протесты, Холмс открывает сейф и находит папку с надписью "Военный Проект" и незаконченное письмо, обрывающееся словами: "Жди меня на мосту...".

Холмсу совершенно ясно, что именно Тришия Фендер шантажировала миссис Браун. Однако, в папке секретных документов не оказалось, очевидно они были похищены.

Холмс приказывает хозяйке дома следовать за ним, берет кэб и везет ее в Скотланд Ярд. Здесь они дожидаются прихода Лестрейда и Холмс приступает к допросу.

Подозреваемая сдается на вопросе об окровавленной одежде, найденной в доме миссис Браун (TELL ME ABOUT THE BLOODSTAINED CLOTHES) и начинает давать показания.

Сразу же открывается, что она вовсе и не Тришия Фендер, а настоящее ее имя - миссис Джонс. Она всегда была близкой подругой миссис Браун и, когда узнала, что та стала жертвой шантажа со стороны Тришии Фендер, убила шантажистку. Ее труп и был найден в библиотеке.

Теперь Холмс может сделать официальное заявление Лестрейду, как представителю власти: "MRS JONES KILLED TRICIA FENDER" ("миссис Джонс убила Тришию Фендер"). Лестрейду нужны неопровержимые улики и Холмс обращается к подозреваемой "TELL LESTRADE WHAT HAPPENED" ("расскажите Лестрейду о том, что произошло").

Если перед этим Холмс тщательно изучил все улики, не пропустил ничего важного, то она во всем сознается инспектору, придет дежурный полисмен и арестует преступницу. Если же это не произошло, значит что-то важное Холмс все-таки упустил.

После ареста миссис Джонс, Холмсу остается разрешить еще одну загадку - для кого же Тришия похитила секретные документы и где они сейчас находятся. После тщательного анализа всех собранных улик, Холмс приходит к выводу, что возможным преступником является Базиль Фиппс.

Из Скотланд Ярда Холмс отправляется на квартиру к Фиппсу. В Лондоне тот проживает в доме на улице Camden Street. Подождав под окнами до 10 часов вечера, можно дождаться, что кто-то изнутри откроет окно. Переодевшись в костюм старика, Холмс проникает в спальню через окно. Быстро осмотрев дом, Холмс остановил особое внимание на библиотеке. Здесь в мусорной корзине он находит клочки какой-то записки. На заднем дворе в мусорном ящике ему впоследствии тоже удается обнаружить порванную записку. Выбравшись из дома, переодевшись и изучив содержимое клочков, Холмс обнаруживает, что это обрывки одного зашифрованного послания. Недолго провозившись с простым шифром, он получает примерно следующее содержание:

"Чертежи у меня. Ваша цена меня устраивает. Прошу сообщить о времени покупки. Смерть миссис Браун вызвала вмешательство полиции. Базиль."

Для Холмса это неопровержимое свидетельство того, что похищенные документы

находятся у Фиппса. С этим он и направляется в Скотланд Ярд, где до 7 часов утра ожидает прихода Лестрейда.

Среда.

Встретив Лестрейда, Холмс сообщает ему, у кого находятся чертежи: "BASIL HAS THE PLANS". Если к этому времени Холмс прочитал содержимое зашифрованной записки и обнаружил пустую папку из-под секретных документов, то Лестрейд предложит ему захватить преступников в момент передачи документов.

Холмс полагает, что сейчас Базиль Фиппс ожидает ответа от неизвестного сообщника и возвращается к дому на Camden Street.

В 9:50 появляется мальчик-посыльный и приносит Фиппсу записку. Холмс должен добыть ее любой ценой. Прокравшись к окну, он заглядывает внутрь комнаты. (LOOK THROUGH WINDOW). Если ничего интересного он не видит, наблюдение надо продолжать (команда повторяется). Наконец, где-то в 10:06 ему удается разглядеть горящий клочок бумаги. Теперь все решает скорость. Быстро влезть в окно. Схватить горящий лист и вылезти обратно. Маскировочный костюм теперь ему уже ни к чему.

Позже ему удается разобрать на обгоревшем листе зашифрованный текст, но на этот раз шифр уже другой. Кроме того, текст, за исключением имени адресата, написан задом наперед. Текст гласит:

"Базиль! Я покупаю чертежи. Буду в два тридцать на дороге Old Mill Road около Лезерхэда." Подписано инициалами H.W.

Быстро вернувшись в Скотланд Ярд, Холмс сообщает о том, что ему удалось узнать. Он отвечает на вопрос Лестрейда о том, где состоится передача секретных документов и они бросаются за преступниками.

Прибыв в 1:30 в Лизерхэд, Холмс выходит из поезда раньше Лестрейда и немедленно идет на главную улицу городка. Здесь уже ждет полицейский кэб. Задача Холмса успеть расположиться в кэбе до того, как его займут Лестрейд с помощником, иначе Холмсу просто не хватит места.

Лестрейд велит кэбмену ехать на Old Mill Road. Они приезжают туда в 3:13, но это уже поздно. Базиль только что уехал. Лестрейд приказывет ехать обратно и в 4:47 они снова возвращаются на главную улицу. Здесь им удается заметить Базиля и рядом с ним немецкого агента. Преследование приводит Холмса на платформу номер 2, где злоумышленники вскакивают в отходящий поезд. Холмс же должен найти другой путь в Лондон. Сказав Лестрейду: "За мной!", (FOLLOW ME), он спешит в полицейский кэб. Уотсон также следует за ними. Холмс велит кэбмену ехать на King Cross Road.

Если все делать достаточно быстро, то Холмс имеет шанс увидеть, как Базиль и агент усаживаются в кэб. Удается и услышать, как Базиль приказывает кэбмену отправляться на Buckingham Palace Road.

Холмс мгновенно соображает, что соответствующая станция подземки - это Виктория (Victoria). Сыщики прыгают в поезд на платформе King Cross, который идет к Victoria, где и выходят.

Через некоторое время появляются преследуемые. Заметив Холмса, Базиль понимает, что он попался и пытается застрелить Холмса. Жизнь ему спасает доктор Уотсон (если он рядом). В последнюю секунду ему удается оттолкнуть Холмса в сторону.

Лестрейд производит арест.

Если Холмс сделает все это, то конечно он станет Величайшим сыщиком всего мира.

Так заканчивается одно из наиболее загадочный и интригующих расследований, которое вел когда-либо Шерлок Холмс. Почему оно не было до сих пор обнародовано, остается загадкой.

* * *

Мы напечатали эти заметки не только для тех, кто увлекается программой SHERLOCK.

Они относятся ко всем, кто любит адвентюрные игры, но пока не может похвастаться особыми достижениями. Пусть действия Холмса послужат примером того, что можно требовать от хорошей программы этого жанра. А для тех, кто еще не начал работу с адвентюрными программами, мы только укажем, что все изложенное здесь представляет кратчайший путь к победе, но и на 10% не исчерпывает всех возможных вариантов развития событий.

Мы также рекомендуем начинающим изучить цикл статей "ADVENTURE LESSONS" в номерах ZX-РЕВЮ 1991 года.

НАШ КОНКУРС

В номере 1-2 ZX-РЕВЮ за этот год, мы объявили конкурс по программе ELITE на самый рискованный маршрут.

Напомним условия: в одной из галактик надо облететь 20 планет, не побывав на одной планете дважды. Подсчет уровня опасности избранного маршрута производится по следующим критериям:

- анархическая планета - 5 очков;
- феодальная планета - 4 очка;
- любая другая - 1 очко.

Призы в конкурсе: пять первых мест получают ксерокопию повести DARK WHEEL на английском языке. Повесть написана по мотивам программы ELITE.

Как обычно, конкурс вызвал большой интерес среди наших читателей. По нему поступило около 100 писем с предложениями самого "крутого маршрута". Абсолютное большинство читателей пришли к выводу, что самым злачным местом во Вселенной является Галактика N4. Так получилось, что те читатели, которые пытались проложить маршрут по другим галактикам, не добрали очков и остались вне числа призеров.

Наилучший результат, достигнутый нашими пилотами - 100 баллов достигнут Ивановым А.И. из Читы. Мы, правда, должны сказать, что были еще такие же достижения, но получались они с нарушением правил. Например, когда пилот прокладывал маршрут по двум и более галактикам, пользуясь гипердрайвом. Такие решения мы не рассматривали.

На втором месте с результатом 92 балла оказалось сразу 12 человек.

Вот как выглядит наш Зал Славы:

Дзреев К.К. (Ростов на Дону)
Довженко В.П. (Киев)
Жаров Р.Н. (Херсон)
Игнатов А. (Новосибирск)
Коротков О.Е., Шилов А.В. (Ростов на Дону)
Минаков Р.С. (Калининград)
Минеев А.В. (Владивосток)
Мясников Г.Л. (Сыктывкар)
Радзевич А.А. (Нальчик)
Сергиенко Т.П. (Волжский, Волгоградской обл.)
Тошев В.В. (Норильск)
Хохлов Д.В. (Санкт-Петербург)

Нет числа разбитым пиратским кораблям, теперь они долго будут усеивать четвертую галактику, а перед нами новая проблема: как разделить 5 призов между 13 победителями (у нас просто больше нет ни одного лишнего экземпляра).

Мы приняли такое решение:

Иванову А. И. отправляем повесть "DARK WHEEL", путем жеребьевки она же отправляется Игнатову А., Короткову О.Е., Радзевичу А.А., Тошеву В.В.

Остальным же победителям мы в качестве завоеванного приза вышлем имеющиеся несколько экземпляров книги Яна Логана и Фрэнка О'Хары "Полный дисассемблер ПЗУ" тоже на английском языке. Эта книга является библией всякого настоящего "синклериста" и мы надеемся, что победители не будут огорчены произведенной заменой.

А что же касается самой повести "DARK WHEEL", то сотни наших читателей спрашивают, почему бы нам не перевести ее на русский язык и не опубликовать по частям в ZX-РЕВЮ? Действительно, такое решение выглядит наиболее рациональным и мы просто упустили его из виду, не ожидая что она вызовет массовый интерес. Так мы и сделаем в ближайших выпусках.

Сегодня мы предлагаем вниманию наших коммерческих представителей для проведения маркетингового исследования новую информационно-поисковую систему "DBP".

DATA BASE PROCESSOR - процессор баз данных.

Вы помните, мы представляли в N1-2 за этот год систему многоцелевого назначения "РЕГИСТРАТУРА". "ПРОЦЕССОР" обладает всеми теми же свойствами, но имеет ряд мощных дополнительный возможностей, делающих его самостоятельным коммерческим продуктом. Предполагается, что наибольший эффект он даст при совместном использовании с "РЕГИСТРАТУРОЙ".

I. НАЗНАЧЕНИЕ СИСТЕМЫ

Как и система "РЕГИСТРАТУРА", система DBF предназначена для ведения учета и проведения обработки любой информации, необходимой на Вашем предприятии (в Вашем учреждении) Например:

- регистрация входящих и исходящих документов;
- регистрация входящих и исходящих товаров;
- регистрация пациентов в лечебном учреждении;
- регистрация клиентуры и заказчиков; - учет движения материалов на складе;
- и многое, многое другое.

Система может удовлетворить потребности отдела кадров, планово-финансового отдела, отдела соцкультбыта, различных административных, муниципальных и хозяйственных служб и т.п., кроме бухгалтерии.

II. ВОЗМОЖНОСТИ СИСТЕМЫ

Не останавливаясь на всех возможностях системы "РЕГИСТРАТУРА", перечисленных в N1-2 (стр. 44), мы укажем на дополнительные инструментальные возможности, предоставляемые ПРОЦЕССОРОМ при создании и модификации баз данных.

- теперь нет необходимости при заказе базы приводить список необходимых полей с указанием их типа и размера. ПРОЦЕССОР позволяет пользователю самостоятельно создавать базы данных с любой необходимой ему информацией. Пользователь имеет возможность задать те поля информации, которые ему необходимы в данный момент.

- пользователь теперь имеет возможность изменять структуру даже ранее созданной базы. Можно в любое время создать новые поля, удалить ненужные, произвести замену. Вставка новых полей может происходить и в середину готовой и заполненной базы.

- удобным дополнением является поиск повторяющихся 8 записей по одному или нескольким полям.

- введена функция поиска и замены информации (по аналогии с текстовыми редакторами).

- при копировании и печати информации добавлена возможность пометки блока записей из заданного интервала.

- введен режим подведения суммарного итога по заданнному числовому полю.

III СТРАТЕГИЯ ЭКСПЛУАТАЦИИ

Система DBF полностью заменяет ранее выпущенную систему "РЕГИСТРАТУРА", но вследствие своих расширенных возможностей, требующих специальных знании при подготовке структуры баз данных, отличается:

а) более высокой ценой;
б) большей сложностью в освоении неподготовленным пользователем.

Поэтому наиболее целесообразно применять эти системы совместно так, если на предприятии несколько подразделений нуждаются в информационно-поисковых системах, целесообразно в каждом иметь систему "РЕГИСТРАТУРА", настроенную на потребности данного подразделения и одну центральную систему DBF для первичной подготовки и настройки информационных файлов для этих подразделений с возможностью последующего обслуживания и модифицирования.

IV. КОМПЛЕКТ ПОСТАВКИ

Система поставляется на одной дискете 5,25" (MS DOS, 360 K).
К системе прилагается подробная инструкция по эксплуатации.

V. СРОК ИСПОЛНЕНИЯ ЗАКАЗА.

Срок исполнения - 1 - 2 недели после поступления средств на наш р/с и письма-заказа.

VI. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К АППАРАТНО-ПРОГРАММНОМУ ОКРУЖЕНИЮ

1. Полная аппаратно-программная совместимость с IBM PC XT/AT. Надежность функционирования на отечественных модификациях не гарантируется и не обсуждается.
2. Наличие "жесткого" диска ("Винчестера") стандартного объема.
3. Дисковод гибких дисков 5,25" или 3,5"".
1. Операционная система - MS DOS не ниже 3.20.
5. Русификация компьютера в стандарте ГОСТ (кодировка альтернативная).
6. Требования к монитору - не специфицируются. Желательно - EGA.
7. Требования к принтеру - совместимость со стандартом EPSON.

VII. ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА

Гарантийным свидетельством при поставке программного продукта является картонный альбом, в который вложены дискеты с указанной на нем датой продажи. При его отсутствии поставка выполняется с заверенным гарантийным талоном.

Гарантиями обеспечивается: - бесплатная замена поставочных дисков, неработоспособных в состоянии поставки (в течение месяца после поставки);

- замена с минимальной оплатой при выходе программ из строя по вине пользователя (механическое или электромагнитное повреждение, поражение вирусом на машине пользователя и т.п.) или по истечении месяца после поставки. Минимальная оплата не превышает стоимость дисков + 5% текущей стоимости программного обеспечения + стоимость почтово-транспортных расходов и согласовывается с потребителем.

VIII. ПОРЯДОК ОФОРМЛЕНИЯ ЗАКАЗА.

а) направить в наш адрес письмо-заказ с указанием необходимого программного продукта и количества копии. Приложить копию платежного поручения.

Наш адрес: 121019, Москва, Г-19, а/я 16, "ИНФОРКОМ".

б) произвести предварительную оплату платежным поручением на наш р/с: N 500461778 во Фрунзенском коммерческом банке г. Москвы.

Стоимость системы на период июль-сентябрь 1992г. - 12500 рублей + 28%

"ЗЕЛЕНЫЙ ПАКЕТ" ДИСТРИБУТОРА

О том, что такое "зеленый пакет" Вы можете подробно прочесть в N11-12 "ZX-РЕВЮ" за 1991г.

Стоимость "зеленого пакета" по системе DBP составляет для частных лиц:

Рабочая версия программы - 10% * 12500 =	1250
Дискета -	100
Почтовые расходы -	15

Итого:	1365 рублей

Уточняем, что высокая стоимость дискеты вызвана тем, что нашим дистрибуторам поставляются дискеты особо высокого качества с защитным тефлоновым покрытием, имеющие особый представительский вид и высокую коммерческую стоимость.

Напоминаем, что затраты дистрибутора на "зеленый пакет" являются только залогом и возвращаются по требованию. Активно работающим дистрибуторам затраты возвращаются при выплате комиссионных и далее такие пакеты предоставляются бесплатно.



МКП "ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

Письмо читателя.

" Привет, ИНФОРКОМ!

Всегда с удовольствием читаю "РЕВЮ" от первой до последней страницы, но простите, что-то в последнее время не пойму, что Вы затеяли. Вся страна объявляет доподписки на газеты и журналы и требует доплаты, а вы упорно стоите на своем. И слепому ясно, что на те девяносто рублей, которые стоила подписка в прошлом году Вам не продержаться. Вам там в Москве легко играть в благородненьких. Мы, мол не такие, как все. А вот нам здесь сермяжным что прикажете делать, когда все Ваше дело квакнет? Вы-то выкрутитесь, у вас ИБМ-ы всякие и прочие дела, а мне куда деваться с моим "Ленинградом", который я пол-года паял, если этих самых ИБМ во всем городе полторы штуки и никто не знает, как к ним подойти?

Знаете, ребята, давайте кончать это дело. Вы бы хоть с народом посоветовались. Может, людям и не жалко денег за "РЕВЮ". А вот если вы задумали это дело свернуть, как невыгодное, то имейте в виду, Родина Вам это не простит".

Вот такое пришло письмо. Мы не приводим имени автора, их несколько сотен, писем с таким содержанием. Мы выбрали не самое мягкое и не самое жесткое и сохранили авторский стиль. Такие письма приходят уже несколько месяцев. И привели мы его вовсе не для того, чтобы как это часто делают "по просьбе населения" отмочить какую-нибудь гадость. Ничего подобного. Никаких доподписок как не было, так и не будет. Мы твердо стоим на том, что если издатель принял от людей деньги авансом, когда червонцы были большими, то и отработать он их должен полностью. Это его трудности, если он не сумел или не успел их использовать по назначению своевременно - для закупки бумаги, техники, оплаты перспективных разработок.

Пусть доподписку просят те, кто все проел и промотал вместо того, чтобы организовать дело с перспективой.

Вместе с тем, это письмо явилось и поводом для глубоких размышлений. И мы сделали три генеральных вывода:

1. Наша работа нужна, ее ценят и если мы ее свернем, то Родина нам не простит.

2. Если наша работа нужна и мы ее не будем развивать, то она свернется сама и Родина нам не простит.

3. Если мы будем развивать свою работу так, как мы это могли бы и хотели, то нас загонит в гроб нынешнее положение с почтой и типографиями. Мы превратимся в некое подобие давно почившей организации "Книга-почтой", скончаемся и Родина нам не простит.

Замкнутый круг и разорвать его можно только если среди Вас, уважаемые читатели,

найдутся предприимчивые люди, способные подхватить наше дело и продвинуть в своих регионах.

Мы сосредоточиваем все усилия на подготовке новых материалов, а вопросы печати и распространения по регионам отдаем Вам на условиях лицензирования.

У нас лежат без движения еще десятки тысяч страниц необработанной информации и, если мы, как и сейчас, будем 90% сил и энергии тратить на почту и печать, то они так никогда до Вас и не дойдут.

Мы давно начали готовить развернутую серию книг по "Спектруму" и приглашаем включиться в дело их издания и распространения всех, кто имеет для этого физические и материальные возможности. Сейчас количество "синклеристов" в стране уже составляет несколько миллионов, а мы едва можем обслужить десятые доли процента. Только совместными усилиями мы дойдем до каждого.

Итак, назовем условно готовящуюся серию "Библиотека ИНФОРКОМа". Оговоримся сразу, что это будут книги совершенно новые, оригинальные, а не перепечатки того, что и так циркулирует по рынкам всей страны. Объем каждой книги - 200... 220 страниц. Что в эту серию войдет:

Самая ближайшая задача - сериал, посвященный графике "Спектрума". Пока он разрабатывается в четырех томах.

Т. 1 "Элементарная графика".

Он уже готов и за его издание и распространение можете приниматься хоть сейчас. В порядке презентации одна глава из этого тома напечатана в сегодняшнем номере "РЕВЮ".

Т. 2 - "Прикладная графика". Здесь рассмотрены вопросы деловой, научной, игровой, векторной, трехмерной, теневой графики и пр. и пр.

Т. 3 - "Динамическая графика".

Имеется в виду использование динамической графики для создания мультипликационных и анимационных эффектов в Ваших программах,

Т. 4 - "Дизайн Ваших программ". Не повторяя материалов первых трех томов, здесь рассмотрены вопросы применения специальных приемов и эффектов для получения полноценных программ. Эта книга является кульминацией всего того, о чем было сказано в первых трех томах. Один отрывок из этого готовящегося сейчас тома также представлен на страницах сегодняшнего выпуска.

Над томами 2,3,4 сейчас идет работа. По окончании работы над ними в перспективе планируется решить вопрос о целесообразности выпуска пятого тома "Справочника по машинной графике", но это уже будем смотреть по обстоятельствам. Здесь есть проблемы.

Во всяком случае, первые четыре тома сейчас готовятся в предположении, что никакого справочника нет и все, что нужно для понимания материала, в них имеется.

Кроме графического сериала, идет подготовка и других. Во всяком случае, для первых 15-17 томов материал уже собран и они находятся в работе. В потенциале если все пойдет хорошо, можно будет довести общее количество книг до 22-25.

Вторым крупным готовящимся сериалом будут тома, посвященные вопросам самостоятельной разработки программ - обучающих и прикладных, а также адвентюрных и аркадных игр.

Третий сериал чисто игровой, содержит готовые программы для самостоятельного набора. Отрывок из книги, посвященной настольным играм, представлен в данном выпуске (см. программу "Крибедж").

Кроме тома настольных игр готовятся тома стратегических, аркадных, адвентюрных и

обучающих программ.

Мы особенно рады, что сможем удовлетворить значительно выросший интерес читателей к адвентюрным программам, дело в том, что печать этих программ сопряжена с одним большим неудобством. Ведь человек, набравший текст программы на компьютере уже не сможет в нее полноценно играть, т. к. во время набора узнает все хитрости и загадки. Разработанная нами система "скрэмблинга" исходного текста позволяет уйти от этого недостатка, т. к. теперь набор текстовых сообщений идет в закодированном виде.

Наши предложения.

Теперь рассмотрим, как мы представляем себе организацию работы.

Заинтересованным организациям и предпринимателям мы передаем дискеты для IBM-совместимых машин в формате MS DOS, на которых записан готовый отформатированный текст книги. Если по ходу текста есть рисунки, то они содержатся на тех же дискетах отдельно в формате .PCX файлов.

Таким образом, мы передаем будущему издателю готовый оригинал-макет книги, распечатать который дело нескольких часов, вклеить рисунки и можно его отдавать в типографию.

Наши условия:

1. Издатель не вносит никаких изменений в исходный текст без согласования с авторами.

2. Желая дополнить текст какими-либо материалами, например рекламными, издатель прилагает их только в конце книги и от своего имени.

3. Издатель обязуется не предпринимать никаких целенаправленных усилий для издания или распространения книги за пределами того региона, который он представляет.

В свою очередь "ИНФОРКОМ" гарантирует, что лицензия на издание и распространение данной книги в этом регионе не будет предоставлена другой организации. Лицензия выдается на каждую книгу (том) отдельно по мере ее готовности.

В то же время, "ИНФОРКОМ" вставляет за собой право индивидуального обслуживания своих клиентов единичными экземплярами, в каком бы регионе они не проживали.

4. Издатель сам определяет тираж и предполагаемую розничную цену издания.

5. Финансовые расчеты с " ИНФОРКОМом" выполняются в два этапа.

Издатель оплачивает от 3% до 11% от предполагаемого объема реализации авторскому коллективу в виде авторского гонорара, после чего получает готовый текст на дискетах и лицензию от "ИНФОРКОМа" на работу в своем регионе.

После выхода тиража книги издатель бесплатно передает "ИНФОРКОМу от 100 до 400 авторских экземпляров готовой книги.

Мы ждем реакции с мест на наши предложения. Представленный разброс в процентах авторского гонорара (от 3 до 11) и в поставках готовой продукции (от 100 до 400) является объектом конкретный прямых переговоров и действует по принципу "чем больше планируемый объем реализации, тем меньше процент" и "чем больше тираж, тем больше пакет авторских экземпляров".

Пишите, телеграфируйте. Вам будет незамедлительно дан телефон для прямых контактов.

В условиях развала бывшего Союза, коллапса единой банковской системы и нарушения нормальной работы почты мы полагаем, что наше предложение вызовет должную заинтересованность. К примеру, у нас просто нет никаких других способов обслужить республики Прибалтики и ряд других регионов.

В соответствии с распределением любителей ПК "Спектрум" мы составили проспект лицензирования по регионам. Отныне в будущих номерах "ZX-РЕВЮ" мы будем регулярно освещать ход дел в этой области.

Ниже мы приводим карту регионирования. Регионы расположены в порядке убывания критерия перспективности (оценка субъективная). Москва и область не представлены, на них лицензия не выдается, здесь мы будем работать сами.

С.-Петербург и обл.
Екатеринбург и обл.
Красноярский край.
Казахстан
Тюменская обл.
Челябинск и обл.
Кемеровская обл.
Киев, Чернигов и обл.
Донецк и обл.
Минск и обл.
Брест и обл.
Мурманск и обл.
Новосибирск и обл.
Днепропетровск и обл.
Крым
Самара и обл.
Нижний Новгород и обл.
Респ. Саха (Якутия)
Хабаровский край
Пермь и обл.
Харьков и обл.
Ростов и обл.
Краснодарский кр.
Рязань, Тамбов, Воронеж, Липецк и области
Приморский край (Владивосток)
Иркутская обл.
Винница, Черновцы, Тернополь, Ивано-Франковск, Хмельницкий и области
Латвия
Респ. Коми
Орел, Курск, Белгород
Ярославль, Иваново, Кострома и области
Архангельск и обл.
Томск и обл.
Алтайский край
Магаданская обл.
Башкортостан (Уфа)
Татарстан (Казань)
Чувашия (Чебоксары)
Львов, Ужгород
Удмуртия (Ижевск)
Ставропольский край
Запорожье
Одесса и обл.
Литва
Узбекистан
Камчатская обл.
Саратов и обл.
Волгоград, Астрахань
Луганск
Омск

Чита
Тверь
Респ. Молдова
Респ. Карелия
Гродно и обл.
Могилев и обл.
Калининградская обл.
Калужская обл.
Полтавская обл.
Кировоградская обл.
Эстония
Херсон и обл.
Николаев и обл.
Ульяновская обл.
Брянская обл.
Вологодская обл.

Конечно, здесь представлена далеко не вся география. Это только крупнейшие центры, в которых наше дело развито. Мы с радостью установим связи с предпринимателями и из других регионов.

И последнее. Обращаем внимание крупнейших производителей компьютеров, что приложение хороших книг к Вашей продукции может очень сильно способствовать повышению ее конкурентоспособности.

Желаем успехов!

Ваш "ИНФОРКОМ".

BETA BASIC

Начало см. стр. 3, 47, 91

34. LIST FORMAT число.

Эта команда использует сразу два ключевых слова БЕЙСИКа и предназначена для того, чтобы управлять форматом листинга Вашей программы, т.е. с ее помощью можно получить распечатку программы в удобном для восприятия виде.

Команда имеет несколько режимов, которые задаются параметром. Начальное состояние после загрузки Бета-Бейсика - LIST FORMAT 0.

LIST FORMAT 0.

Эта команда дает распечатку, похожую на ту, которую мы получаем в стандартном БЕЙСИКе, но есть незначительное отличие, заключающееся в том, что если строка программы длиннее, чем строка экрана, то перенос ее на вторую экранную строку выполняется со сдвигом. Таким образом, левые пять столбцов экрана содержат только номера программных строк и текст становится более разборчивым и удобочитаемым.

LIST FORMAT 1.

Распечатка по этой команде делается таким образом, что каждый оператор печатается с новой строки. Более того, некоторые из операторов (см. ниже) печатаются со смещением вправо на одну позицию, что позволяет распечатывать программу "лесенкой", так как это принято в языках, поддерживающих структурное программирование, например "ПАСКАЛЬ", "СИ" и др.

LIST FORMAT 2.

Действие то же, что и для LIST FORMAT 1, но автоматическое смешение для некоторых операторов выполняется не на одну, а на две позиции вправо.

LIST FORMAT 3.

Действие то же, что и для LIST FORMAT 0, но печать программы выполняется без номеров строк.

LIST FORMAT 4.

Действие то же, что и для LIST FORMAT 1, но печать программы выполняется без номеров строк.

LIST FORMAT 5.

Действие то же, что и для LIST FORMAT 2, но без номеров строк.

Для глаза приятнее воспринимать смещение "лесенкой" на две позиции, но поскольку у нас невелика ширина экранной строки - всего 32 символа, то ее может оказаться недостаточным и поэтому введены режимы 1 и 4.

Появление в тексте программы следующих операторов вызывают автоматический сдвиг листинга:

DEF PROC, DO, FOR - сдвигают все последующие операторы вправо на один или на два символа до тех пор, пока не встретятся соответствующие им END PROC, LOOP или NEXT.

Операторы IF, ON ERROR и ON сдвигают все прочие операторы своей программной строки вправо.

Операторы ELSE и EXIT IF отменяют сдвиг текущего оператора на одну или две позиции.

Если Вам захочется, чтобы оператор после ELSE или THEN печатался с новой строки, то Вы можете после них поставить двоеточие.

Пример действия команд LIST FORMAT 0 и LIST FORMAT 2 показан ниже.

```
100 DEF PROC primer
110 FOR n=1 TO 10: PRINT "primer": NEXT n
120 DO: PRINT a$: INPUT b$
130 PRINT "abc": LOOP
140 IF x=1 THEN: PRINT "yes": PAUSE 50: ELSE: PRINT "no"
150 END PROC
```

```
100 DEF PROC primer
110   FOR n=1 TO 10
          PRINT "primer"
      NEXT n
120   DO
          PRINT a$
          INPUT b$
130       PRINT "abc"
      LOOP
140   IF x=1 THEN
          PRINT "yes"
          PAUSE 50
      ELSE
          PRINT "no"
150   END PROC
```

Если Вы очень привыкли к стандартному БЕЙСИКУ, то первое впечатление будет, что второй листинг выглядит несколько странно, но немного практики и вы почувствуете, насколько он удобнее.

35. LIST PROC имя процедуры.

Команда позволяет распечатать не всю программу, а только процедуру, которую Вы задали именем, например LIST PROC box.

На практике эта команда очень часто используется в длинных программах. Тогда Вам не надо помнить, в каком месте программы расположена Ваша процедура.

Может быть, Вам захочется хранить имя процедуры в строковой переменной. Команда LIST PROC не сможет тогда ее обработать напрямую и требуется обходной прием:

```
10 INPUT a$: KEYIN "LIST PROC"+a$
```

Две системные переменные содержат номера первой и последней строк процедуры, заданной в LIST PROC. Их адреса 23635 и 57358, соответственно. Работа с ними иногда может быть очень полезной и ее удобно выполнять, используя функцию DPEEK.

36. LIST REF.

REF - ключевое слово на клавише "SHIFT"+"7".

См. также REF.

LIST REF - дает список номеров помеченных строк. Меткой может быть имя переменной, число или набор символов. LIST REF тесно связана с командой REF и описана более подробно в соответствующем разделе, который желательно предварительно прочитать.

Если в некоей программной строке метка существует не один раз, то при печати списка этот номер строки появится многократно

Например:

```
10 FOR n=1 TO 10: PRINT n
20 NEXT n
```

Для такой программы команда LIST REF n даст следующий список:

```
10
10
20
```

Чтобы направить список на принтер, Вы можете использовать команду LLIST REF или:

```
LIST #(номер потока) REF метка
```

37. LOCAL переменная <,переменная><,переменная>...

Клавиша: "SHIFT" + "3"

См. также главу 3 "Процедуры" и раздел "DEF PROC".

Команда LOCAL позволяет создавать специальные переменные, которые существуют только внутри заданных процедур. (Параметры процедур автоматически имеют статус локальных и потому не нуждаются в объявлении их оператором LOCAL.) Использование команды LOCAL для создания локальных переменных в процедурах имеет то преимущество, что исключает образование помех, вызванных тем, что процедура может изменить содержимое какой либо переменной, относящейся к главной программе. Благодаря этой команде процедура может иметь переменные x и a$ и изменять их как угодно, не изменяя переменные x и a$ в главной программе.

Команда LOCAL может быть использована только в теле объявленной процедуры, иначе система выдаст сообщение об ошибке "Missing DEF PROC". Самое удобное для нее место - после объявления процедуры, сразу за DEF PROC. Тогда при чтении распечатки сразу становится ясно, какие переменные в процедуре используются. По желанию Вы можете иметь в процедуре несколько команд LOCAL.

Если переменные, объявленные как LOCAL, до этого объявления в программе уже существовали, то они "прячутся". На самом деле они, конечно существуют в области программных переменных и команда CLEAR их не уничтожит, но с точки зрения процедуры их как бы нет. В процедуре теперь эти имена могут использоваться снова. В конце работы процедуры все переменные, объявленные как LOCAL, будут уничтожены и исходные значения (если они были спрятаны) будут восстановлены. Вот простой пример.

```
10 LET n=1
20 test: REM чтобы выйти из курсора "K" воспользуйтесь ведущим пробелом.
30 PRINT n
40 STOP
100 DEF PROC test
110 LOCAL n
120 LET n=999
130 PRINT n
140 END PROC
```

Запустите этот пример и убедитесь, что n в процедуре равно 999, а вне ее n равно 1. Если вы опустите строку 120, то n не будет существовать внутри процедур, а если Вы опустите строку 10, то n не будет существовать в главной программе.

Если наша процедура test будет внутри себя вызывать некую субпроцедуру (например из строки 125), то для нее переменная n со значением 999 будет глобальной, для этой субпроцедуры процедура test является как бы главной программой.

Все переменные, доступные в процедуре являются глобальными по отношению к процедурам, вызываемым из нее. Чтобы "спрятать" эти переменные в нижележащих процедурах. Вы можете и в них использовать оператор LOCAL или использовать их в списке параметров.

В отличие от многих прочих диалектов БЕЙСИКа, БЕТА-БЕЙСИК поддерживает локальные массивы. Если, к примеру, в главной программе Вы используете a$, то в процедуре командой LOCAL a$ все строковые массивы a$ или переменные a$ будут локальными и не повлияют на переменные главной программы. Если Вы хотите числовой массив сделать локальным, то после его имени надо использовать круглые скобки. Например, LOCAL b(). Это "спрячет" любой существующий массив b(). После этого Вы можете в процедуре создать свой новый локальный массив b() того размера, который Вам нужен.

38. LOOP

или LOOP WHILE условие

или LOOP UNTIL условие

клавиша: L

См. также DO, EXIT IF.

LOOP - это нижняя граница цикла DO-LOOP (см. DO). Команда LOOP, заданная сама

по себе вызывает возвращение исполнения программы к соответствующему оператору DO.

LOOP может употребляться вместе с кодификаторами WHILE и UNTIL, которые обеспечивают возврат в начало цикла по условию.

LOOP WHILE условие - обеспечивает возврат назад, к вершине цикла только в том случае, если <условие> справедливо, в противном случае выполняются операторы и строки, стоящие за LOOP.

LOOP UNTIL условие имеет противоположное действие. Если указанное <условие> справедливо, возврат к вершине цикла не выполняется, а выполняется, когда оно ложно.

Если Вы используете в программе оператор LOOP без соответствующего DO, то получите сообщение об ошибке: T,"LOOP without DO".

39. MERGE

См. также DEFAULT = устройство. Эта команда имеет дополнительную возможность, которую использовать смогут только немногие владельцы микродрайва. Она позволяет загружать без автозапуска те программы, которые выполнены автостартующими. С ленты это можно делать и так.

40. MOVE

Эта команда тоже развивает возможности владельцев микродрайва. Она теперь позволяет перемещать не только блоки данных, но и программы, машинный код и массивы.

Чтобы перенести файл "test" с драйва 1 на драйв 2, делайте так: MOVE "m"; 1; "test" TO "m"; 2; "test"

Команда сработает, независимо от физической природы файла. Хоть MOVE и выглядит удобной командой, на практике большие файлы быстрее перемещаются через SAVE/LOAD.

41. ON

Клавиша: O

БЕТА-БЕЙСИК обеспечивает две различные формы оператора ON.

Вы можете задать после ON список число или выражение, которое определит, к какой строке будет выполняться переход по оператору GO TO или GO SUB.

Это довольно традиционный оператор для различных диалектов БЕЙСИКа, хотя и довольно архаичный в эпоху широкого применения процедур.

Вторая форма позволяет не только избрать строку, но и оператор.

Пример. Первая форма.

```
GO TO ON число, номер строки, номер строки, номер строки...
```

или

```
GO SUB ON число, номер строки, номер строки, номер строки...
```

(Более стандартной выглядела бы форма ON число GO TO строка..., но клавиатурная система "Спектрума" делает ее усложненной).

Обычно при программировании на "Спектруме" такой многопозиционный переключатель организуют так:

```
10 INPUT choice:
      GO TO choice*100+100
```

Но с использованием ON это можно делать более гибко, т.к. номера строк перехода не должны быть числами одного заданного ряда.

```
10 INPUT choice:
      GO TO ON choice: 90,135,60,40
20 PRINT "Enter 1 to 4": GO TO 10
```

В этом примере будет выполнен переход к строке 90, если choice равно 1, к строке 135, если choice равно 2, к строке 60, если choice равно 3 и т.д. Если choice не входит в диапазон от 1 до 4, то никакого перехода не делается, а выполняется следующей оператор или программная строка, а она предоставляет пользователю возможность повторить свой

выбор. В этом примере вы можете заменить INPUT на GET и тогда получите элегантную технику для написания программ, управляемых от меню.

Вторая форма.

```
ON число: оператор: оператор:...
```

Эта форма позволяет исполнить тот или иной оператор из списка, в зависимости от того, какое значение принимает параметр <число>.

Из списка операторов исполняется только указанный, после чего программа переходит к исполнению следующей строки. Если параметр <число> больше, чем количество операторов в строке, то сразу выполняется переход к следующей строке. Ниже приведен пример использования этой команды в ответ на ввод от пользователя числа 1,2 или 4. Если он введет число 3, то никакой реакции не произойдет и программа продолжится в естественном порядке. Это обеспечено тем, что третий оператор в строке 20 - пустой.

```
10 INPUT x
20 ON x:  PRINT "one":
          PRINT "two"::
          PRINT "four"
30 GO TO 10
```

Совсем необязательно, чтобы операторы в строке ON были одного типа. Ниже показан пример использования "смешанных" конструкций, содержащих и вызов процедуры и GO SUB и PRINT.

```
10 DO
20    GET number
      ON number
          GO SUB 100
          sound
          GO SUB 200
          PRINT "bye"
30 LOOP
```

Использование ON со списком процедур - наиболее современная, рациональная и удобочитаемая форма.

42. ON ERROR номер строки

или

ON ERROR: оператор: оператор:...

Клавиша: N.

Для оператора ON ERROR возможны две формы записи. Первая форма задает номер строки, к которой происходит переход, если происходит прерывание работы по ошибке. Остальные операторы в строке ON ERROR в этом случае не имеют к нему специального отношения.

Во второй форме после ошибки начинается выполнение операторов, содержащихся в данной строке.

За ошибку считается любая ошибка из числа перечисленных в инструкции к "Спектруму" или из числа приведенных в приложении к инструкции по работе с языком "БЕТА-БЕЙСИК 3.0", т.е. все сообщения Бейсика, кроме сообщений:

```
0: "OK"
9: "STOP statement"
```

Этот режим можно отключить оператором ON ERROR 0, но он также отключается автоматически при работе процедуры обработки ошибки и вновь включается после возврата в главную программу. (Ей приходится отключаться, иначе был бы конфуз, когда при обработке собственной ошибки она вызывала бы саму себя и вновь сталкивалась с той же ошибкой.)

Этой процедуре доступны три специальные переменные, которые могут быть очень и очень полезны: LINO, STAT и ERROR. Все это не ключевые слова и набираются по буквам.

LINO и STAT - номер строки, в которой произошла ошибка и номер оператора в строке.

ERROR - номер кода ошибки, когда мы закончим печать инструкции мы дадим коды всех ошибок.

Вы можете пользоваться этими переменными в своих целях, но делать это надо до того, как будет активизирована процедура ON ERROR, потому как при ее активизации, как и при активизации режима TRACE (см. ниже), эти переменные могут быть перезатерты.

Не рекомендуем Вам все возможные ошибки обрабатывать через ON ERROR. Желательно все, за исключением одного-двух сообщений обрабатывать естественным путем, иначе при отладке программы может быть трудно разобраться с тем, что в ней происходит. Во всяком случае Вы ведь не предполагаете, что Ваши программы будут изобиловать многочисленными ошибками. В нижеприведенном примере программа печатает точки на экране, а процедура обработки ошибок отсекает те, которые выходят за его пределы.

Первая форма выглядит так

```
100 ON ERROR 5000
110 FOR n=1 TO 10:
      INPUT "x coord "; x;
      "y coord "; y
120 PLOT x,y: NEXT n
.........................
4990 STOP
5000 IF error=11 AND lino=120
      THEN RETURN: ELSE POP:
      CONTINUE
```

Обратите внимание: Оператор STOP стоит в программе для того, чтобы предотвратить случайное исполнение процедуры обработки ошибки. Значения LINO и ERROR проверяются потому, что сообщение "Integer out of range" является широко распространенным и появляется во многих случаях. Возврат из процедуры обработки ошибки выполняется к оператору, следующему за тем, который вызвал ошибку, поэтому возврат выполняется к оператору NEXT n. Если номер строки или код ошибки были не те, то выполняется оператор CONTINUE. В результате этого подозрительный оператор выполняется еще раз (кроме случая "BREAK into program") и, поскольку CONTINUE не задействовал ON ERROR, то теперь обработка ошибки пойдет стандартным путем. Оператор POP очищает стек от адреса возврата в главную программу. Если этого не сделать, то нормальная работа стека будет нарушена.

Вторая форма выглядит так:

```
100 ON ERROR
      IF error=11 AND lino=120
         THEN RETURN
      ELSE
         POP
         CONTINUE
```

Есть одна "ошибка", которую необходимо обрабатывать путем, отличным от прочих. Это "BREAK into program". Поскольку ON ERROR отключается, когда выполняется переход GO SUB к процедуре обработки ошибки, то обычное действие BREAK будет состоять в том, что Ваша процедура остановится после исполнения первого же оператора, поскольку Вы еще не успели отпустить клавишу BREAK. Поэтому, если Вы хотите, чтобы по нажатию BREAK были предприняты какие-то действия и ваша процедура обработки ошибок работала бы, то вам необходимо ввести паузу в первом операторе процедуры обработки ошибки, достаточную для того, чтобы пользователь освободил клавишу BREAK. Для этой цели неплохо может служить оператор BEEP. Если вы не хотите, чтобы генерируемый звук был слышен, используйте те частоты, которые не слышны для человеческого уха. Например:

```
100 ON ERROR 5000
110 PRINT "round and ";:
      PAUSE 10: GO TO 110
...........................
4990 STOP
5000 IF error = 21
      THEN BEEP 1.69:
```

```
BORDER RND*7:
RETURN: ELSE POP:
CONTINUE
```

43. OVER 2.

См. также GET, PLOT.

Кроме 0 и 1 OVER теперь может работать с параметром 2. OVER 2 имеет то действие, что те символы или рисунки, которые печатаются командой PLOT <строковая переменная>, добавляются к тому, что уже есть на экране, а не затирает имеющееся изображение и не инвертируют общие точки. Пиксел приобретает цвет INK, если он был INK или он становится INK от нового рисунка. Таким образом, OVER 2 аналогичен слиянию изображений по логике OR, в то время как OVER 1 работает по логике XOR. (Различие в их логике будет описано в разделе "функции").

44. PLOT X,Y <;строка>

См. также: GET, OVER 2, CSIZE, управляющие коды.

Как видите, Бета-БЕЙСИК допускает выполнение PLOT не только для точек, но и для символьных строк. Это могут быть обычные символьные строки или экранные блоки, снятые с экрана с помощью GET и сохраненные в памяти, как строковые переменные.

Координаты, выступающие параметрами оператора PLOT имеют отношение к левому верхнему углу той символьной последовательности, которая помещается на экран. Могут использоваться и все обычные прочие квалификаторы оператора PLOT, такие, как INVERSE, OVER, INK и т.п. Если печатаемая строковая последовательность выходит за пределы правого нижнего угла экрана, то она появляется в левом верхнем углу. Если какая-то часть помещаемого на экран символа выходит за его пределы, выдается сообщение об ошибке "Integer out of range". В то же время, нижние 7 пикселов изображаемых символов могут выходить за нижнюю границу и в этом случае они изображаются в системном окне экрана.

Координаты позиции PLOT, которые используются в качестве стартовых для работы оператора DRAW, не нарушаются при использовании команды PLOT со строковой переменной.

Изменяя координаты позиции PLOT для символа, Вы можете добиться гораздо более плавного эффекта мультипликации, чем это возможно в обычном БЕЙСИКе при использовании оператора PRINT AT.

```
100 FOR x=16 TO 224: PLOT x,x/2; "<>": NEXT x
```

Попробуйте ввести в цикл параметр STEP 2 или 3 или более. Скорость будет выше, а эффект не совсем тот. Поскольку рисунок "<>" имеет вокруг себя поле цвета PAPER шириной по крайней мере в один пиксел, то при движении по экрану он будет сам себя стирать, если перемещение делать на один пиксел за шаг. Некоторые буквы, например "T" имеют включенные пикселы цвета INK, подходящие к самой кромке знакоместа, поэтому существуют такие направления движения символа, при котором он не стирается, а оставляет след на экране. Если Вы сами конструируете себе символьный набор, то не помешает делать его так, чтобы со всех сторон символа оставалось поле шириной в один пиксел.

Вы можете увеличивать или уменьшать размеры помещенных на экран символов, графики, блоков и т.п. с использованием команды CSIZE (см. соответствующий раздел). Команда CSIZE должна непосредственно следовать за PLOT. Например:

```
PLOT CSIZE 32;INK 2: 100,88;"HI!"
```

В печатаемую символьную строку Вы можете включить управляющие коды CHR$ 8 - CHR$ 11. Тогда Вы сможете получать гораздо более сложные геометрические построения.

Возможность помещать строки на экран командой PLOT очень полезна для печати графиков и диаграмм. Во-первых, здесь возможна более высокая точность, а во-вторых можно пользоваться той же координатной системой, которой пользуются графические функции, например DRAW, CIRCLE.

45. POKE адрес, строка

БЕТА-БЕЙСИК дает возможность не только выполнять POKE для чисел, но и для символьных строк, что в совокупности с функцией MEMORY$ дает возможность скоростных манипуляций с большими массивами памяти. Надо, правда, отметить, что если нерасчетливый POKE какого-либо числа может вывести из строя программу, то для строк это становится еще более критичным.

Рассмотрим пример:

```
10 LET screen=16364
20 POKE screen, STRING$ (6114, "U")
```

О функции STRING$ мы будем говорить позже, в разделе "Функции". Здесь мы заполняем экранную память кодом буквы "U", но воспринимается это не как код, а как двоичное число 01010101, что изображает на экране полосы.

Следующий пример демонстрирует копирование начальной области ПЗУ в файл экранных атрибутов.

```
10 LET attr = 22528
20 POKE attr, MEMORY$()(1 TO 704)
```

Теперь давайте нарисуем на экране что-нибудь простое, сохраним изображение в строковой переменной и затем восстановим его на экране, используя POKE.

```
10 CIRCLE 128,88,70
20 FILL 128,88
30 LET a$ = MEMORY$() (16384 TO 23295): REM весь экран
40 CLS: PRINT "нажмите любую клавишу": PAUSE 0
50 POKE 16384,a$
```

В памяти компьютера могут одновременно храниться несколько таких картинок, вы можете оперативно менять их местами и по одной выбрасывать на экран. Если Вам надо больше картинок, вы можете одну треть экрана считывать в строковую переменную. Если вас интересует не только черно-белая информация, но и атрибуты цвета, то и треть файла атрибутов Вы тоже можете сбрасывать в такую же переменную.

Для хранения черно-белой информации экранных сегментов вы может пользоваться следующими командами:

Верх:

```
LET a$=MEMORY$()(16384 TO 18431)
```

Середина:

```
LET a$=MEMORY$()(18432 TO 20479)
```

Низ:

```
LET a$=MEMORY$()(20480 TO 22527)
```

У компьютера достаточно памяти даже для того, чтобы исполнить реальный мультфильм путем использования команды POKE в сегмент экрана. Чтобы хранить данные в памяти, можно использовать массив DIM a$(10,2048).

Конечно, потенциальных возможностей у манипуляций с большими объемами памяти гораздо больше, чем просто обслуживание экранной памяти. Вы можете, например очистить большой объем верхней памяти и сохранить там целую программу, а потом вызывать ее опять. Так можно обеспечить одновременное присутствие в памяти компьютера сразу нескольких программ.

```
CLEAR 33900
10 POKE 34000, MEMORY$() (22552 TO 33800)
20 REM остальные строки этой программы.
```

Теперь Вы можете сделать NEW и удалить Вашу программу, но ее копия останется в верхней памяти выше границы RAMTOP, установленной оператором CLEAR.

Можно опять вызвать эту программу в работу:

```
POKE 33552, MEMORY$() (34000 TO 44248)
```

Эту вызывающую команду можно "подвесить" на клавишу, определяемую пользователем и, поскольку такие определения тоже хранятся выше уровня RAMTOP, то оно будет защищено от разрушения командой CLEAR.

```
DEF KEY "j": POKE 23552, MEMORY$() (34000 TO 44248)
```

В тот момент, как программа будет восстановлена, она продолжит работу с того места, в котором она была "спрятана", т.к. вместе с программой отгружались и восстанавливались все ее системные переменные.

Итак, как видите, БЕТА-БЕЙСИК дает Вам несложный механизм организации виртуального диска (RAM-диска) для работы сразу с несколькими программами.

И еще одна идея для использования POKE, заключающаяся в возможности сохранения машинного кода вместе с БЕЙСИК-программой. Присвойте этому блоку кодов имя символьной переменной и выгрузите БЕЙСИК на ленту таким образом, чтобы при последующей загрузке POKE, находящийся в строке автостарта, автоматически перебрасывал этот код на нужное место памяти и дальше запускайте его. Это делается значительно быстрее, чем раздельная загрузка БЕЙСИК-программы и блока машинных кодов. Для того, чтобы освободить место под блок машинного кода и не потерять при этом БЕЙСИК-переменные, воспользуйтесь командой CLEAR.

46. POP <числовая переменная>

Клавиша: Q

Команда POP удаляет последний адрес со стека, обеспечивавшего правильную работу команд GO SUB, DO-LOOP, PROC. Если при этом Вы используете параметр <числовая переменная>, то номер строки, к которой должен был бы быть исполнен переход, если бы вы не сняли его со стека, запоминается в этой переменной.

Команда POP может позволить Вам завершать работу в подпрограммах, процедурах и циклах не естественным путем и при этом закупорки стека не произойдет, если выходя в неположенном месте, например из цикла, Вы снимете адрес естественного возврата со стека. Команда POP без параметра просто удалит этот адрес со стека, а если Вы используете ее с параметром, например с переменной loc, то этот адрес еще останется в вашей переменной и, может быть, он Вам впоследствии для чего-нибудь пригодится. Может быть, по ходу программы Вы примете решение о том, что POP был сделан неправильно и надо все-таки перейти туда, куда должна была возвращать Ваша процедура. В этом случае знание loc позволит Вам сделать GO TO loc+1, хотя надо отметить, что это все же не вполне то же самое, что и естественный RETURN. Дело в том, что адрес, откуда Вы прошли в подпрограмму, запомнен на стеке и после RETURN Вы возвращаетесь туда, откуда уходили. Возвращаетесь либо на следующую строку, либо на следующий оператор в той же строке, если он есть. Переход же по GO TO loc+1 не может вернуть Вас к следующему оператору, а только к следующей строке.

Пример:

```
100 GO SUB 500
110 STOP
500 POP loc
510 PRINT "Подпрограмма вызывалась из строки "; loc
520 GO TO loc
```

Если Вы в строке 520 поставите

```
520 RETURN
```

то получите сообщение "RETURN without GOSUB", поскольку к этому моменту на стеке уже не будет никакого адреса, ведь он был снят командой POP.

Вызов команды POP, когда на стеке нет данных, дает сообщение об ошибке V, "No POP data".

47. PROC имя <параметр><,параметр><,параметр>...

Клавиша: 2

См. также Главу 3 "Процедуры", а также раздел DEF PROC.

Для того, чтобы вызвать процедуру, в версии 3.0 нет необходимости давать ключевое слово PROC и оно сохранено главным образом для того, чтобы обеспечить совместимость с предыдущими версиями БЕТА-БЕЙСИКа. Вы можете обращаться к процедуре по ее имени без этого ключевого слова, для чего имя вводятся после отключения K-режима вводом

ведущих пробелов или вызовом режима KEYWORDS 4.

С другой стороны, слово PROC используется, например, в команде LIST PROC (см.).

48. READ LINE строковая переменная <,строковая переменная>...

См. также Главу 3 "Процедуры".

Команда состоит из двух ключевых слов и позволяет работать без кавычек со списком данных, которые обычно должны иметь кавычки. Например:

```
100 DATA dog, rat, fish, frog, z$,12,"?*+"
110 READ LINE a$ 120 PRINT a$: GO TO 110
```

Оператор DATA ограничивает типы данных, которые можно хранить таким способом, поскольку допустимы только полноценные выражения. В строке 100 "cat" может быть числовой переменной, но "?*+" требует наличия кавычек. Оба типа можно было бы объединить, добавив:

```
115 IF a$(1)=CHR$ 34 THEN LET a$=VAL$ a$
```

READ LIKE позволяет сделать строки DATA более удобными для чтения, хотя главное назначение этой команды - сделать возможным ввод строковых данных без кавычек.

49. REF метка

Клавиша: SHIFT + "7"

Эта команда позволяет выполнить в программе поиск заданной "метки", которая может быть числом, именем переменной или набором символов.

Когда указанная метка найдена, строка, содержащая ее, появляется в системном окне компьютера (в строках редактирования). При этом курсор стоит непосредственно за меткой. Если Вы не хотите вносить изменений в найденную строку. Просто нажмите ENTER. Чтобы найти другие строки, содержащие такую же метку, нажмите ENTER еще раз и так далее, когда поиск будет закончен полностью, появится сообщение "O.K. "

Если вместо нажатия ENTER вы введете какую-либо команду, то компьютер решит, что Вы закончили поиск и теперь, чтобы опять продолжить поиск. Вам надо снова задавать команду REF.

Примеры: (символы, не являющиеся буквой, цифрой и знаком $ показаны, как _.)

```
REF a$    ищется a$
REF count        ищется _count_
REF "count"      ищется count
REF 1            ищется 1 (число)
REF "1"          ищется 1 (символ)
REF 12*4         ищется 12 (число) * 4 (число)
REF (a$)         ищется значение a$, так, например, если a$ = "fish", то ищется "fish", а не
     a$.
REF x            ищется значение x, например, если x=10, то ищется 10(числовая форма).
```

Не играет никакой роли регистр встреченных символов - прописные буквы или строчные.

При поиске переменных необходимое условие, что имя должно начинаться и заканчиваться символами, отличными от буквы или цифры, Это позволяет отличать разные переменные, имеющие общие символы в имени, например:

```
count account counts
```

При поиске чисел поиск идет не по их символьному представлению на экране, а по их числовой интегральной (пятибайтной) форме, которая стоит за всяким числом, появляющимся в любой БЕЙСИК строке. Это тоже позволяет избежать конфузов. Таким образом, внутренние вложения в переменные и числа не мешают правильной работе.

Если же Вы ищете набор символов, заключите его в скобки и он будет найден, даже если является вложением в большую строку.

Если Вы используете строковую переменную при поиске, то ее надо заключить в скобки, чтобы БЕТА-БЕЙСИК мог отличить случай поиска по имени от поиска по значению.

50. REF имя переменной.

Клавиша: SHIFT + "/"

Мы рекомендуем Вам предварительно прочитать главу 3 "Процедуры", прежде чем двигаться дальше.

Здесь ключевое слово REF используется для того, чтобы сообщить процедуре о том, что такие-то и такие-то параметры этой процедуры передаются в виде ссылки, а не по значению.

Это означает, что во время исполнения процедуры соответствующие переменные могут быть временно переименованы в имя, стоящее после REF. Это обеспечивает передачу параметров не только из вышележащей процедуры в нижележащую, но и наоборот снизу вверх. Пассивы всегда должны передаваться, как ссылки. В нижележащем примере REF a$ относится к символьной строке или массиву, а REF b() - относится к числовому массиву.

```
100 DEF PROC crunch REF a$, REF b(), REF OUTPUT
```

51. RENUM <*><начало TO конец> LINE <новое начало> STEP <шаг>.

Клавиша: 7

RENUM обеспечивает очень мощные возможности по перенумерованию блоков программных строк, их перемещению и их копированию.

Просто команда RENUM без параметров перенумерует все строки программы так, что номером первой строки будет 10, а далее все строки будут идти с шагом по 10. Но можно и задать перенумерацию не всей программы, а только заданного блока строк.

RENUM 130 TO 220 - перенумерует все строки из этого диапазона.

RENUM 130 TO - перенумерует все строки, начиная со строки 130 и до конца программы.

RENUM TO 100 - перенумерует строки, начиная с первой программной строки и до строки 120 включительно, за исключением нулевой строки.

Перенумерованный блок будет перемещен в программе на заказанное место, если в памяти для него место имеется, в противном случае может быть выдано сообщение об ошибке: G, "No room for line" ("Для строки нет места").

Команда RENUM* отличается тем, что она выполняет не перемещение строк, а их копирование, т.е. создается новый блок с заданной нумерацией строк, но старый при этом не уничтожается.

Вы можете задать в качестве первой строки не десятую, а любую другую, указав параметр LINE (это ключевое слово). Если Вас не устраивает стандартный шаг нумерации строк в БЕЙСИКе через десять, задайте свой в качестве параметра STEP (ключевое слово). И LINE и STEP могут быть даны или опущены по желанию, но если Вы их даете, то порядок их следования должен быть только таким, как указано в описании команды и не наоборот.

Некоторые примеры:

```
RENUM
RENUM LINE 100 STEP 20
RENUM 100 LINE 300 (перенумеровывается только одна строка)
RENUM 1540 TO LINE 2000
RENUM 100 TO 176 LINE 230 STEP 5
RENUM * 10 TO 100 LINE 500 -копирование блока строк.
```

При перенумерации программных строк все ссылки на строки по их номеру тоже перерабатываются, включая GO TO, GO SUB, RESTORE, RUN, ON, ON ERROR, TRACE, LIST, LLIST, LIKE.

Операторы DELETE и CLOCK - не перерабатываются, Вам придется заняться этим вручную.

RENUM не может также самостоятельно переработать вычисляемые ссылки на номера строк, такие как GO TO ln*100.

После завершения перенумерации все указания на такие подозрительные места будут распечатаны на экране, чтобы Вы могли с ними разобраться, например:

```
Failed at 100:2
Failed at 230:4
```

Все GO TO, GO SUB и т.п., содержащие после себя такие вычисляемые адреса перехода, будут распечатаны, даже если Вы и перенумеровываете лишь малый блок и в ней нет таких операторов, потому что они могут быть в других местах программы и указывать именно на тот блок, который Вы и перенумеровали.

Если Вы желаете направить распечатку таких сложных строк на принтер, воспользуйтесь:

```
OPEN #2, "P": RENUM:
OPEN #2, "S"
```

Примечание: во время работы команда RENUM образует временный буфер в экранной области компьютера, что сопровождается неожиданными помехами на экране. Не обращайте на них внимание, по окончании работы экран будет восстановлен.

52. ROLL код направления <, число><; x, y; ширина, длина >

Клавиша: R

См. также SCROLL

Команда ROLL перемещает изображение, имеющееся на экране или в заданном окне вверх, вниз, влево или вправо. Все, что выходит за пределы окна, тут же появляется с противоположной стороны. Одним словом, команда, в отличие от команды SCROLL, не разрушает содержимое экрана, а только перемещает его.

Как видите, команда ROLL имеет довольно сложный синтаксис, хотя большая часть сопровождающих параметров служит только для того, чтобы задать окно, внутри которого действует ROLL. Если Вы, например, хотите сдвинуть на один пиксел содержимое текущего окна, а таковым обычно является весь экран, если вы не задали иначе, то команда имеет такой простой вид:

```
10 ROLL код направления
```

Если вам надо сдвинуть только черно-белую графику без цветовых атрибутов, то в качестве кода направления нужно задать 5,6,7 или 8 (соответственно это означает влево, вниз, вверх, вправо).

Поскольку команда ROLL вызывает смещение изображения на небольшую величину, самый лучший способ ее применения - внутри циклов.

Нарисуйте (DRAW, CIRCLE) какое-либо несложное, но крупное изображение или просто дайте команду LIST и экран будет заполнен, а затем попробуйте следующие строки:

```
100 FOR d=5 TO 6: FOR p= 1 TO 100
110 ROLL d
120 NEXT p: NEXT d: STOP
```

Можно делать смещение и по диагонали. В этом случае в цикле выполняется последовательность движений вверх, вправо или т. п. Если хотите, чтобы движение шло более быстро (но менее плавно), попробуйте:

```
110 ROLL d, 4
```

Этот параметр указывает на сколько пикселов за один раз должно производиться смещение. Он не должен быть более, чем 256 при горизонтальном движении и чем 177 - при вертикальном. Если параметр не указав, по умолчанию принимается единица. Очевидно, что чем больше это число, тем более значительно будет передвинуто изображение. При вертикальной движении скорость обычно пропорциональна количеству пикселов, передвигаемых за раз. При горизонтальной движении наилучший результат дает шаг в 4 или в 8 пикселов, т. к. в этом случае используются более скоростные команды машинного кода процессора Z-80.

Если Вы хотите передвигать не только чёрно-белую информацию, но и цветовые атрибуты, добавьте к коду направления число 4, а если хотите передвигать только атрибуты, наоборот отнимите 4.

Код	Направление	Действие
1	Влево	Атрибуты
2	Вниз	Атрибуты
3	Вверх	Атрибуты
4	Вправо	Атрибуты

5	Влево	Графика
6	Вниз	Графика
7	Вверх	Графика
8	Вправо	Графика
9	Влево	Граф. + Атр.
10	Вниз	Граф. + Атр.
11	Вверх	Граф. + Атр.
12	Вправо	Граф. + Атр.

Атрибуты можно сдвигать только на 8 пикселов за один шаг и параметр, определяющий шаг смешения здесь игнорируется.

При использовании кодов от 9 до 12 можно добиваться наилучшего соответствия между шагом перемещения черно-белой графики и цветовых атрибутов. Посмотрите, что происходит:

```
10 PRINT AT 10, 10: PAPER 2; "DEMO"
20 ROLL 9: PAUSE 10: GO TO 20
```

Вы увидите, что рассогласование движения атрибутов и графики достигает 4-х пикселов. Теперь попробуйте такой вариант:

```
10 PRINT AT 10, 10; INK 2; " DEMO "
```

Дополнительные пробелы, охватывающие слово "DEMO", обеспечивают то, что оно всегда прикрыто 6-символьной полосой красного цвета. Аналогичный метод работает и для других геометрических форм. Нижеприведенный пример создает цветные круги, каждый из которых окружен защитным кольцом с соответствующей установкой атрибутов.

```
10 LET y=88, r=15
20 FOR n=1 TO 4
30 LET x=n*48+8
40 CIRCLE x,y,r
50 FILL INK n; x, y
60 CIRCLE INK n; INVERSE 1: OVER 1; x, y, r+5
70 NEXT n
80 ROLL 9
90 GO TO 80
```

Если перемещению подлежит не весь экран, а только его часть (окно), то его Вы можете задать, указав координаты x и y левого верхнего угла, ширину и высоту. При этом здесь применяются разные координатные системы. Параметры X, Y и высота окна задаются в пикселах, а ширина окна - в знакоместах. Ее конечно тоже можно было бы задавать в пикселах, но тогда команда получается медленноработающей. Итак, параметр ширины может быть от 1 до 32, а параметр высоты - от 1 до 176.

Что же касается атрибутов, то они перемещаются только с точностью до знакомест, поэтому работая в цвете необходимо четко планировать свои действия.

Вы можете предварительно задать окно командой WINDOW и тогда команда ROLL будет работать с данным окном без необходимости указывать его параметры.

ROLL может эффективно применяться при создании аркадных игр для обеспечения плавных движений героя или ландшафта. Интересный головокружительный эффект может иметь создание сразу нескольких пересекающихся окон, двигающихся в разных направлениях.

```
100 LIST: LIST: LIST
110 LET pixels=4
120 ROLL 5, pixels; 0,175;32,88
130 ROLL 6, pixels; 0,175;16,176
140 ROLL 8, pixels; 0,87;32,88
150 ROLL 7, pixels; 128,175;16,176
160 GO TO 120
```

Поэкспериментируйте с этой программой. Попробуйте задать pixels=1. Попробуйте изменить строку 100 на следующую:

```
100 KEYWORDS 0: PRINT STRING$(704 , " END PROC "): KEYWORDS 1
```

Слова END PROC набирайте не по буквам, а клавишей 3 в графическом режиме. И, наконец, последний пример:

```
200 FOR n=1 TO 7: LIST: NEXT n
```

```
210 FOR m=1 TO 175:
      ROLL 5;0,175;32,m:
      NEXT m
```

(Продолжение в следующем выпуске)

Михайленко В.С.

ЗАЩИТА ПРОГРАММ

Продолжение.
(Начало см. стр. 9-16, 53-60, 97-104).

Итак, Вы осуществили подмену и загрузили Бейсик-файл под видом кодов. Теперь Вашей задачей является просмотреть этот файл и изучить его структуру. Для этого можно использовать приведенную ниже программу Бейсика:

```
1 FOR i=30000 TO (30000+"реальная длина")
2 PRINT i:TAB 7;PEEK i;TAB 11;CHR PEEK i
3 NEXT i
```

В первой строке величина "реальная длина" является своим значением для каждого конкретного случая рассматриваемой вами программы. Она определяет ту область программы, которую Вы желаете просмотреть. (Поскольку мы загрузили Бейсик-файл под видом кодов в область памяти компьютера, начиная с 30000, то это значение является исходной точкой для начала просмотра).

После того, как вы набрали текст программы дампинга и запустили ее командой RUN, то на экране появятся столбцы значений в следующем формате:

Значение ячейки памяти	Содержимое данной ячейки памяти в DEC виде	Символьное представление содержимого ячейки памяти

Чтобы Вам хорошо разбираться в сути текста, появлявшегося на экране, необходимо вспомнить структуру Бейсик-строки. Как известно, ее схематично можно представить в виде:

MM NN текст строки код ENTER (13)

где:

MM - два байта номера строки

NN - два байта длины строки

"Текст строки" почти соответствует исходному, за исключением представления цифровых величин (очень подробно информация о представлениях чисел в ZX SPECTRUM описана в трехтомнике "ИНФОРКОМа" "Первые шаги по программированию в машинных кодах") и завершает строку код ENTER 0BH (13)[1].

Когда перед Вами появится текст, будьте очень внимательны - со временем Вам будет достаточно легко читать его. В первую очередь следите за появлением кода ENTER 0BH (о его появлении свидетельствует перевод позиции печати к началу следующей строки, а также появление в строке дампинга "цифровое содержимое ячейки памяти" значение 13). После этого кода мысленно пропускайте 4 цифры (от них все равно очень мало толку, поскольку значение номера строки Бейсика из них не очевидно - это же касается и длины. Чтобы получить реальное значение, необходимо старший байт умножить на 256 и прибавить к этому значению младший байт) и внимательнейшим образом изучаете структуру Бейсика.

Для того, чтобы сделать текст Бейсик-строк исходного файла более читаемым, можно заменить строку 2 программы дампинга на следующую строку:

```
2 PRINT CHR$ PEEK i;
```

После этого Вы будете иметь картину почти аналогичную листингу, за исключением необычного представления номера строки и числовых значений.

Внимание: работа программы дампинга может быть прервана выдачей какого-либо

---

[1] Код ENTER равен 0Dh (Прим. OCR)

сообщения, например о неправильном цвете:

```
INVALID COLOR
```

или большом целом числе:

```
NUMBER TOO BIG
```

В этом случае необходимо набрать с клавиатуры NEXT i и дампинг продолжится.

После того, как Вы внимательно изучили структуру программы, необходимо точно определить местонахождение подпрограммы в кодах, чтобы не путать ее с последовательностью символов Бейсика. Обычно эта подпрограмма размещается после оператора REM и состоит из самых разнообразных символов.

Когда все это осуществлено, необходимо изменить содержимое этой Бейсик-программы таким образом, чтобы после ее загрузки в компьютер и запуска по команде автостарта она сама останавливалась, например по команде STOP. Это необходимо сделать таким образом, чтобы STOP сработал до блокированных POKES, которые могут не остановить работу программы, а дестабилизировать ее работу, например вызвав зависание или самосброс.

Наиболее разумным с нашей точки зрения является введение оператора STOP вместо первого оператора Бейсика. Это можно осуществить подав команду с клавиатуры

```
POKE 30004, CODE "STOP"
```

Теперь, после загрузки нашего Бейсика и запуска его на выполнение, осуществится останов по команде STOP. Наша задача выполнена. Теперь необходимо записать информацию на кассету. Подадим команду с клавиатуры:

```
SAVE "имя" CODE 30000, реальная длина
```

Однако не торопитесь включать магнитофон и нажимать ENTER. Образующийся после подачи этих команд блок будет содержать хэдер кодов и непосредственно файл кодов, аналогичный файлу Бейсика. Однако, если Вы внимательно разберетесь, то поймете, что хэдер кодов это не что иное, как "Специальный хэдер кодов (2)" т.е. фактически вам записывать его вовсе не обязательно. В данном случае на ленту можно записать только измененный файл кодов, который будет аналогичен файлу Бейсика за исключением внесенных изменений. Обозначим этот измененный файл кодов (2').

Теперь Вам необходимо загрузить измененную программу Бейсика в память компьютера. Подаем с клавиатуры команды: LOAD "" и сначала загружаем исходный хэдер Бейсика (1), после которого загружаем измененный файл кодов (2'). После загрузки программа должна остановиться с сообщением команды STOP.

Примечание: может произойти так, что измененная Бейсик-программа не остановится, а продолжит свою работу, один из возможных вариантов - это неточное внесение изменений. Дело в том, что Бейсик-файл может иметь приблизительно такую структуру:

```
1 REM - подпрограмма в кодах
2 RANDOMIZE USR - или другие команды запуска этой подпрограммы в кодах.
```

Когда Вы вносили изменения, то могли заменить код оператора REM кодом оператора STOP.

Если бы программа автостартовала со строки 1, то она, естественно остановилась бы с выдачей сообщения о выполнении оператора STOP. Однако бывают случаи, когда автозапуск осуществляется со строки 2 и, таким образом, получается, что введенный нами оператор STOP программой не обрабатывается - следовательно остановка не происходит.

Для того, чтобы исправить этот дефект, необходимо снова загрузить исходный Бейсик-файл (1') под видом файла кодов, т.е. вместе с хэдером (2). Однако теперь необходимо вносить изменения более точно, а именно в строку, которая точно обрабатывается интерпретатором Бейсика таким образом, чтобы произошел останов по выполнении оператора "STOP".

Но вот мы добились своей цели - исходная программа загружена в память компьютера без автостарта и мы приблизительно знаем ее структуру, теперь необходимо определить типы защиты, которые применены в данной программе. Здесь возможны, естественно, множество вариантов, но, тем не менее, можно с уверенностью сказать, что наиболее часто встречаются защиты, основанные на применении метода зануления

номеров строк программы, а также связанные с использованием управляющих кодов ZX SPECTRUM для сокрытия листинга исходной программы. Более подробно методы взлома в подобных случаях будут описаны в главе 3.

Теперь рассмотрим достоинства и недостатки данного метода взлома.

Одним из больших достоинств данного метода блокировки автозапуска является его доступность и простота в освоении. В самом деле, здесь Вы не используете никаких непонятных Вам программ и наиболее сложным является понять весь технологический процесс, однако, если Вам это удалось, то можно сказать, что сняты все психологические барьеры.

В то же время, несмотря на кажущуюся простоту, данный метод обладает массой недостатков. В первую очередь, это необходимость серьезной работы с магнитофоном - необходимо очень четко выставлять магнитофон перед запуском магнитной ленты, чтобы у Вас точно грузился тот или иной блок.

Кроме этого, здесь необходимо использовать дополнительное пространство на магнитной ленте для записи переделанных файлов, что не всегда удобно для пользователя.

И все же начинающим хэкерам я рекомендовал бы начинать именно используя этот метод взлома. Несмотря на некоторое неудобство, в некоторых случаях он бывает просто незаменим. Я сам достаточно долгое время работал используя исключительно этот метод и считаю, что благодаря его использованию очень многому научился. Несмотря на то, что методы, которые будут предложены Вашему вниманию в последующих разделах более совершенны и продуманны, они уступают этому методу в главном - там практически все за Вас осуществляет специальная программа и Вы фактически не участвуете во взломе. Здесь же Вы все осуществляете сами вручную и именно этот факт открывает большие перспективы.

2.2 Изменения в хэдере с использованием копировщика COPY-COPY.

Поработав достаточно длительное время, используя технологию, описанную в первой разделе этой главы, я решил усовершенствовать процесс. Основными целями при этом я поставил себе упрощение работы с кассетой и ускорение самого процесса, следует отметить, что достаточно проблематичным является создание новой методики, когда длительное время работал по другой. Но это однообразие в конце концов и помогло.

В этой работе нам понадобится копировщик COPY-COPY. Это достаточно совершенный копировщик, поскольку он имеет ряд возможностей, делающих его незаменимым в данной конкретном случае. Но, кроме всего этого, он является еще и универсальным средством взлома. Именно такое сочетание функций плюс небольшой объем памяти, занимаемый программой, и привели к необычайно широкому ее распространению. (Именно ввиду необычайной его популярности я и привожу ниже описание метода взлома, основанного на его использовании. При этом я надеюсь, что большинство читателей уже имеет в своем архиве этот копировщик, а даже если и не имеют, то ознакомившись с приведенным ниже описанием, приобретут его).

Все команды копировщика являются ключевыми словами компьютера и поэтому не набираются по буквам. Команды требуют завершения нажатием клавиши ENTER.

Приведенная ниже сводка команд разбита на блоки применения. Каждому ключевому СЛОВУ соответствует лишь одна команда копировщика. Но, в зависимости от текста, набранного после ключевого слова, выполняемые операции изменяются. Общим для каждого блока является наличие во всех его командах ключевого слова ZX SPECTRUM.

1. "CAT" - клавиша "C" - просмотр списка имен файлов на экране.

2. "LOAD" - клавиша "J" загрузка файлов в память.

LOAD - загружает программу с очередным номером (имеется в виду нумерация файлов в копировщике)

LOAD N - загружает файл на место N . Если N = 1, то загруженные перед этим файлы теряются и загрузка производится в начало рабочей области (с адреса 23296)

LOAD N TO NN - загружает файлы с номерами от N до NN.

LOAD TO N - загружает файлы от очередного номера до номера N.

LOAD AT NN - загружает файлы с адреса NN, по умолчанию файл с номером 1 загружается с адреса 23296. Можно задать NN = 23040, в этом случае величина рабочей области для загрузки файлов увеличивается до 42496 байтов. По умолчанию величина этой области равна 42240 байтов, очевидно, что NN<23040 задавать нельзя, за исключением случаев загрузки в экранную область (16384).

LOAD (NN - считывает первые NN байтов файла. Это очень удобная функция для получения стандартной копии экранов используя файлы, где загрузка экрана неразрывно сливается с загрузкой программы. Эта функция может очень помочь тем, кто имеет принтер и желает распечатывать красивые картинки. В большинстве программ картинки защищены именно таким способом (RAMBO 2,RAMBO 3, MIG 29 и т. д.) Так, например, команда

```
LOAD (6912)
```

осуществит загрузку данных в формате экрана.

3. "SAVE" - клавиша "S" - сохранение файлов.

SAVE - сохраняет все загруженные файлы без пауз.

SAVE N - сохраняет все файлы, начиная с файла с номером N.

SAVE N TO NN - сохраняет файлы с номерами от N до NN.

SAVE TO N - сохраняет файлы с номерами от 1 до N.

SAVE STEP N - сохраняет все загруженные файлы, между файлами делает паузы в N секунд.

SAVE N TO NN STEP M - сохраняет файлы с номерами от N до NN, где N - номер первого файла; NN -номер последнего файла; M - пауза между файлами в секундах.

4. "VERIFY" клавиша "V" - проверка сохраненных файлов

VERIFY - аналогично SAVE.

VERIFY N TO NN - аналогично SAVE N TO NN.

VERIFY N аналогично SAVE N.

5. "LET" - клавиша "L" - изменение полей заголовка файла,

например:

```
LET 2=AAA,,1
```

Файл с номером 2 будет иметь имя AAA и стартовый адрес 1.

6."LIST" - клавиша "K" распечатка памяти.

LIST (NN) - задает адрес памяти (по умолчанию принимается равным 0) по этой команде выводится 15 байтов памяти, для каждого из которых приводятся:

- адрес памяти;
- десятичное значение байтов;
- десятичное значение двух смежных байтов;
- символьное значение байта.

Для продолжения вывода информации на экран, т.е. просмотра следующих 15 байтов нажмите ENTER.

7. "POKE" - клавиша "O" - изменение десятичного значения байта.

POKE x,NN

- x - адрес;
- NN - значение двух смежных байтов (>255),

POKE X, N

- X - адрес;
- N - десятичное значение байта (<256);

Примечание: Если значение N лежит в диапазоне 256-65535, то считается, что задано значение двух смежных байтов.

8. "USR" - клавиша "U" - вызывает подпрограмму пользователя.

USR X - вызывает подпрограмму в машинных кодах, расположенную по адресу X. Например, если вы желаете полностью перезапустить систему вашего компьютера, наберите USR 0.

9. "RETURN" - клавиша "7"

- возврат в МОНИТОР, инициализируются системные переменные и таблица каналов, однако полный сброс не выполняется.

10. "COPY" - клавиша "Z"

- осуществляет перевод программы в специальный режим для копирования файлов без заголовков длиной до 49056 байт.

После выдачи команды программа загружает файл в память, а затем, по нажатию клавиши "CAPS SHIFT" выгружает ее необходимое число раз. Повторная загрузка возможна только, если остается не менее 200 байт свободной памяти.

COPY NN - данная команда осуществляет копирование файлов, длиной до 49153 байта, копирование выполняется только один раз.

В заключение рассмотрения работы копировщика приводим условное обозначения типов файлов, используемых при его работе.

P - программа

B - вычислительный код (BYTES)

A - числовой массив

$ - символьный массив

2.2.2 Изменение хэдера для блокировки автозапуска.

В главе 1 мы с Вами достаточно подробно рассмотрели структуру хэдера. Напомним лишь, что именно в хэдере Бейсика задается параметр автостарта (т.е. его наличие или отсутствие и номер строки автозапуска при наличии такового). А поскольку мы используем в своей работе такое универсальное средство, как COPY-COPY, то можем изменить эти параметры вплоть до ликвидации автозапуска программы. Рассмотрим более подробно этот процесс.

Как Вам уже вероятно известно, байты 15 и 16 хэдера интерпретируются по-разному. В заголовках программ, написанных на Бейсике, эти байты содержат номер строки, с которой запускается программа - т.е. номер строки автостарта. Если же программа была записана без опции LINE и после считывания не запускается автоматически, то значение числа, содержащегося в этих двух байтах больше 32767. Как видим, одним из способов нейтрализации самозапускающихся программ, является замена этих двух байтов на число, большее 32767. Осуществить это нам поможет программа COPY-COPY.

Для этой цели загрузим копировщик и считаем необходимый нам заголовок с ленты. После этого и будем собственно осуществлять изменения. COPY-COPY настолько универсальная вещь, что изменения можно производить двумя способами. Первый способ основан на изменении встроенной функции LET, а второй использует оператор LIST для просмотра и POKE для непосредственного изменения содержимого ячеек памяти. Рассмотрим более подробно каждый из этих методов.

Метод первый.

Для того, чтобы использовать возможности функции LET, необходимо иметь представление, в каком формате она задается. Как Вам уже вероятно известно, данная функция в общем виде может быть представлена, как:

LET = имя программы, длина программы, номер строки автостарта, прочие

параметры.

Примечания: здесь рассмотрено применение функции LET копировщика COPY-COPY для изменения параметров Бейсик-хэдера. Для других типов хэдера общий вид будет несколько иным.

Подобная структура общего вида команды LET говорит о том, что если мы хотим изменить параметры хэдера следующего в списке копировщика под номером N, то мы должны набрать соответствующую команду LET, после которой через запятую набирается имя программы, длина программы, номер строки автостарта и т.д. Если же мы не хотим изменять все параметры хэдера, то нам необходимо соблюдать прежний порядок набора команды, только вместо параметров, которые мы желаем оставить неизменными, ничего не набираем, оставляя при этом необходимый контингент запятых, например: (исходный хэдер загружен под номером 1 и мы желаем только исключить автозапуск программы)

```
LET 1=..32768
```

После того, как Вы введете эту команду в копировщике COPY-COPY, то исходный хэдер не будет автоматически запускать исходную программу.

Метод второй.

Основан на получении дампинга хэдера путем встроенных функций COPY-COPY с последующим изменением содержимого ячеек памяти.

Для начала получим дампинг хэдера в том формате, который выдает копировщик.

Для того, чтобы вам было легко сориентироваться, напомним, что после 10 байтов названия идут 2 байта длины блока программы, после которых следует 2 байта, которые характеризуют автозапуск программы на Бейсике, именно эти байты и необходимо изменять для того, чтобы выполнить поставленную задачу. Следует помнить, что в этой паре байтов сначала идет старший байт, а потом младший.

Необходимо напомнить читателю, что для получения дампинга необходимо подать команду LIST 23296. (Если исходный хэдер идет первым - в противном случае необходимо сделать так, чтобы он шел первым).

Для изменения содержимого ячеек памяти необходимо использовать встроенную команду POKE. Ее использование полностью аналогично использованию данной команды в Бейсике ZX SPECTRUM.

Данные два метода изменений в программе COPY-COPY полностью альтернативны и взаимозаменяемы. Первый метод несколько более прост в использовании, однако применение второго метода позволяет вам непосредственно изучить структуру хэдера, что бывает иногда необходимо при детальном исследовании какой-либо конкретной программы.

После того, как Вам удалось изменить хэдер и создать необходимую конфигурацию, блокирующую автозапуск, необходимо выгрузить заголовок на магнитную ленту. Делается это с помощью опции копировщика SAVE.

Теперь, если Вы загрузите Вашу исходную программу вместе с измененным хэдером, то Вам удастся достаточно легко изучить ее структуру данной программы.

Приведенный в этой главе метод взлома является несколько более совершенным, в сравнении с методом, описанным в разделе 2.2.1. Однако, он все еще имеет ряд недостатков, и в первую очередь наиболее неприятным является тот факт, что Вам все еще приходится использовать место на кассете. А во-вторых, не все пользователи имеют копировщик COPY-COPY.

Устранению всех этих недостатков способствует использование при взломе специальной программы, которая будет описана в следующей статье.

2.3 Универсальный метод взлома с использованием специального программного обеспечения.

Как Вы уже вероятно догадались, использование методов описанных в этой главе, сопряжено с некоторыми сложностями и неудобствами. Одним из наиболее существенных является необходимость манипуляций с магнитофоном для записи промежуточных файлов.

Метод, который описан в этом разделе, лишен этого и других недостатков.

Он основан на использовании специальной программы, благодаря которой нам удастся блокировать автозапуск исходной программы. Если ввести в память компьютера эту программу и запустить ее, то после запуска программа начинает ждать первую программу на Бейсике, находящуюся на ленте. Она считывает ее аналогично команде LOAD, однако после загрузки не позволяет программе запуститься - выводит сообщение "0 OK". Кроме этого, данная программа выводит информацию, с какой строки считанная программа должна стартовать.

Теперь, когда Вы получили всю необходимую информацию, можно достаточно быстро изучить структуру Бейсик-файла исходной программы.

Универсальная программа для блокировки автозапуска.

```
 1 FOR i=60000 TO 60025: READ A: POKE N,A: 2 NEXT i
 3 RANDOMIZE USR 60000
 4 DATA 1, 34, 0, 247, 213, 221, 225, 253, 54, 56, 1, 221,54, 1,225, 205,29,7,42, 66, 92, 34,
      69, 92, 207, 255
```

Ниже приведен дисассемблер программы в кодах, которая формируется в данном случае с использованием блока DATA.

```
10     ORG 60000
20     LD BC,34
30     RST 48
40     PUSH DE
50     POP IX
60     LD (IX+58),1
70     LD (IX+1),255
80     CALL 1821
90     LD HL,(23618)
100    LD (23621),HL
110    RST 08
120    DEFB 255
```

Глава 3. Методика просмотра Бейсик - программ.

3.1 Просмотр строк, защищенных управляющими кодами.

Информации, которою читатель получил, ознакомившись с предыдущими главами, достаточно для блокировки автозапуска любой Бейсик-программы к ZX SPECTRUM. Но, преодолев первичную защиту, Вы сталкиваетесь со вторым барьером, порой куда более сложным - защитные управляющие коды (подробно рассмотрены в главе 2 т. 1), а также со встроенными процедурами в машинных кодах (рассмотрены в первой главе т. 1).

Введение в компьютерную программу управляющих кодов может преследовать самые разнообразные цели. Одной из них является создание красочной цветовой гаммы и оптимального расположения информации на экране компьютера при минимальном количестве расходуемой памяти. Это используется в некоторых Бейсик-программах, однако, наибольшее распространение управляющие коды получили в системах защиты, где они кроме вышеописанных свойств приобретают еще функции элементов, препятствующих просмотру содержимого программы рядовым пользователем. Здесь мы рассмотрим некоторые ситуации, возникающие при исследовании программ, содержащих управляющие символы, а также методику блокировки управляющих кодов защиты.

Наиболее часто в компьютерных программах к ZX SPECTRUM используются:

CHR$ 8 BACKSPACE

CHR$ 16 INK CONTROL

CHR$ 17 PAPER CONTROL

Именно методику блокировки этих управляющих кодов мы с Вами и рассмотрим.

Для начала коротко об аспектах применения данных управляющих кодов.

BACKSPACE - "курсор влево" служит для забивания предыдущего символа. Именно этот управляющий код генерируется специальными процедурами компьютера для перевода

курсора влево. В защите программ он имеет несколько иное назначение! Благодаря ему удается скрывать ключевые слова, символы, а также элементы слов. В основном это используется для придания эффекта "солидной защиты" и достаточно редко применение данного управляющего кода связано с дезинформацией. По части введения в заблуждение, а также сокрытия наиболее уязвимой программной информации наибольшее распространение получили управляющие коды INC CONTROL и PAPER CONTROL. Именно их применение во многих случаях скрывает от вас подлинный текст программы, поэтому знание принципов блокировки этих управляющих кодов просто "жизненно необходимо".

При разработке технологии применения блокировки данного типа защиты я руководствовался интересами пользователя. В самом деле, почему пользователь должен что-либо делать, если эти функции можно возложить на компьютер.

Это привело к созданию универсальной программы, применение которой позволит избавить текст исходной программы от управляющих символов. Эта программа является более расширенной по своим возможностям в сравнении с аналогичной программой, рассмотренной в главе 1.

Ниже приведен ее листинг.

```
9990 REM ПРОГРАММИСТ МИХАЙЛЕНКО ВАДИМ МЕНСК МРТИ 1991
9991 PAPER 7:INK 0: BORDER 7:CLS
9992 FOR i=23758 TO 65000
9993 IF PEEK i = 13 THEN IF PEEK (i+1) = 39 AND PEEK(i+2)=6 THEN LIST: STOP
9994 IF PEEK i = 13 THEN LET i=i+4
9995 IF PEEK i = 16 THEN POKE(i+1),0: LET i=i + 2
9996 IF PEEK i = 17 THEN POKE(i+1),7: LET i=i+2
9997 IF PEEK i = 8 THEN POKE i,32 9998 NEXT i
```

Примечание ИНФОРКОМа:

У нас есть небольшое замечание, которое мы при редактировании не внесли в листинг, поскольку он защищен авторской строкой 9990.

Суть его в том, что коды 15, 17, 13, 8 и др. могут появляться в БЕЙСИК-строках и не быть управляющими кодами. Вы знаете, что в БЕЙСИКе после обычного посимвольного представления чисел идет код CHR 14 (NUMBER), после которого то же число представляется в скрытой пятибайтной форме (интегральная форма действительных чисел). Так вот, в этих пяти байтах могут быть любые числа, в том числе и те, которые программа может принять за управляющий код. Этот случай, в принципе надо обходить, например добавив в строке 9994 в ее конце после двоеточия:

```
: IF PEEK i=14 THEN LET i=i+5
```

Отметим, что это упущение никак не влияет на получение листинга со снятыми управляющими кодами, и мы упоминаем об этом только в образовательных целях.

Действует программа следующим образом. После того, как были установлены цвета символов, фона и бордюра, в цикле идет анализ Бейсик-строк. Строка 9993 следит за тем, чтобы не обрабатывалась программа блокировки управляющих кодов. Здесь фиксируется конец предыдущей строки Бейсика и проверяется, не имеет ли следующая строка номер 9990 (Вот почему наличие строки с таким номером обязательно). После того, как эта строка обнаружена (следовательно, вся предыдущая Бейсик-программа уже подверглась обработке) - программа блокировки распечатывает на экране текст исходной программы таким, каким он предстает без управляющих кодов. После того, как это сделано, программа останавливается оператором STOP.

Строка с номером 9994 имеет двойное предназначение. Т.к. всякая строка Бейсика оканчивается кодом ENTER - 0BH, то следовательно мы можем определить окончание Бейсик-строки. Кроме того, известно, что первые 4 символа в строке - это соответствующее кодовое представление номера и длины строки. Номер строки нашей программы может содержать одним из кодов число, равное искомому управляющему коду, но поскольку Бейсик-интерпретатором данная последовательность обрабатывается именно как номер и длина, то изменение значений данных ячеек памяти было бы ошибкой. Поэтому мы не анализируем данные 4 байта. Кроме этого, данный алгоритм несколько ускоряет работу

программы.

Строки 9995-9997 осуществляют поиск управляющих кодов и осуществляют все необходимые изменения. В соответствии с установленными в начале программы (см. строка 9991) значениями цвета осуществляется принудительная установка INK CONTROL в черный цвет, а PAPER CONTROL в белый.

В случае же обнаружения управляющего кода BACKSPACE осуществляется принудительная замена его на код пробела - 32.

Несмотря на всю свою привлекательность, основой которой является доступность для понимания, она обладает рядом существенных недостатков, которые затрудняют ее применение на практике. В частности, это низкое быстродействие и сложность внедрения данной программы-резидента в некоторые типы исходных программ, с которых необходимо снять защиту (бывает, что строки с номерами программы-резидента уже задействованы в исходной программе). Кроме этого, данный резидент проверяет не все типы управляющих кодов (а в качестве защиты от листинга может быть использован практически любой из них). Если же мы дополним резидент еще рядом операций по обезвреживанию всех управляющих кодов, то ее объем значительно увеличится, что повлечет за собой увеличение времени работы в несколько раз. Наиболее радикальным средством для увеличения быстродействия является программирование в машинных кодах, именно в этой области вы сможете получить максимальную скорость работы компьютера. К тому же, при специальной системе программирования эти типы программ неприхотливы к месту свободной оперативной памяти, в котором их размещают. Поэтому, если у Вас имеется возможность, старайтесь всегда переводить свои алгоритмы на язык Ассемблера. Это не только ускорит работу Ваших программ на порядок, но еще и поможет вам на практике изучить действительные принципы работы компьютера (процессора Z80).

Вашему вниманию предлагается программа блокировки защиты из управляющих кодов, написанная на языке Ассемблера. Разумеется, я понимаю, что не все читатели знакомы с данным языком программирования, поэтому ниже описаны принципы ее работы достаточно детально. (Тем, кто не знает, как приступить к подобному типу программ, рекомендую трехтомник "Первые шаги в машинных кодах" - ИНФОРКОМ, 1990).

```
Programming by Mihailenko Vadim.
All rights reserved. Mensk 1991.
Mihailenko Vadim driver system
for "EDITAS-48" files. Special
for "INFORCOM" corporation.

10                  ORG 62030
20 ;                -------------------
30 ;                PARAMETR
40                  LD HL,200
50                  LD BC, (23635)
60 ;                -------------------
70 ;                BREAKCONTROL
60      VOZVR       CALL 8020
90                  JR NC,BREAK
100                 LD A, (BC)
110 ;               -------------------
120 ;               ANALIZ
130                 CP 13
140                 JR Z,STROK
150                 CP 16
160                 JR Z,INK
170                 CP 17
180                 JR Z,PAPER
190                 CP 8
200                 JR Z,BACK
210 ;               -------------------
220 ;               BLOKIROVKA
230 ;               KODOV
```

```
240                CP 18
250                JR Z,BACK
260                CP 19
270                JR Z,BACK
260                CP 20
290                JR Z,BACK
300                CP 21
310                JR Z,BACK
320                CP 22
330                JR Z,BACK
340                CP 23
350                JR Z,BACK
360 ;              --------------------------
370 ;              NEXT PARAMETR
380                DEC HL
390                INC BC
400 ;              --------------------------
410 ;              ENDCONTROL
420                LD A,H
430                CP 0
440                JR Z,ZERO
450                JR VOZVR
460 ;              --------------------------
470 ;              SUBROUTINES
480 ;              END CONTROL
490 ;              ZERO L
500   ZERO         LD A,L
510                CP 0
520                RET Z
530                JR VOZVR
540 ;              --------------------------
550 ;              13TH CONTROL
560   STROK        DEC HL
570                DEC HL
580                DEC HL
590                DEC HL
600                INC BC
610                INC BC
620                INC BC
630                INC BC
640                JR VOZVR
650 ;              -----------------------
660 ;              INK CONTROL
670   INK          DEC HL
690                LD A, 0
700                LD (BC),A
710                JR VOZVR
720 ;              ------------------------
750 ;              PAP.CONTROL
740   PAPER        DEC HL
750                INC BC
760                LD A,7
770                LD (BC),A
780                JR VOZVR
790 ;              ------------------------
800 ;              CODE - 32
810                BACK LD A,32
820                LD (BC),A
630                JR VOZVR
840 ;              ------------------------
850 ;              WHEN BREAK
660 ;              THEN RESTART
870   BREAK        RST 8
880                DEFB 20
890                END
```

Данная подпрограмма в машинных кодах представляет собой универсальный инструмент для снятия защиты, состоящей из управляющих кодов. Эта программа является логическим продолжением программы для блокировки автозапуска (описана в третьем разделе предыдущей главы). Вместе они представляют собой универсальное средство для просмотра содержания любого типа Бейсик-программ. (Методика объединения данных программ будет описана ниже). Кроме того, как самостоятельная программная единица, она позволяет размещать себя в любом месте оперативной памяти ZX SPECTRUM. Это намного расширяет спектр ее возможных применений.

Приведенная ранее Бейсик-программа поможет Вам лучше понять алгоритм. Фактически - это более модернизированный аналог.

Предполагается, что эта программа будет сформирована с адреса 62030, оставляя предыдущие 25 байтов для программы блокировки автозапуска с целью совместного применения.

Первый блок PARAMETR задает параметры работы программы. Здесь в регистр HL заносится длина обрабатываемой программы (в моем случае она равна 200, но это очень легко можно будет изменить, о том, как это осуществить, будет описано ниже). В регистр BC заносится содержимое системной переменной PROG. Это необходимо для того, чтобы определить начальную точку работы программы. Поскольку мы собирается корректировать содержимое Бейсик-файла, то нам естественно необходимо знать адрес, с которого он начинается, а именно на него и указывает содержимое переменной PROG.

Следующий блок программы - это анализ в цикле содержимого текущей ячейки памяти, которое заносится в аккумулятор. Это очень похоже на принцип работы Бейсик-программы аналогичного назначения. Фактически эта часть программы, работающая в цикле, состоит из пяти небольших блоков: BREAKCONTROL, ANALIZ, BLOKIROVKA KODOV, NEXT PARAMETRES и ENDCONTROL.

BREAKCONTROL осуществляет проверку нажатия клавиши BREAK. Для этой цели используется встроенная процедура, вызов которой осуществляется командой CALL. В случае же нажатия клавиши BREAK (это определяется по состоянию флага C регистра F) осуществляется переход на подпрограмму BREAK, которая осуществляет рестарт с выдачей сообщения о нажатии клавиши BREAK.

В этом же блоке осуществляется загрузка в аккумулятор текущего содержимого ячейки памяти, адрес которой определяется содержимым регистра BC.

Блок ANALIZ ведет последовательную проверку содержимого аккумулятора, используя функцию сравнения Z80 – CP. Наиболее приоритетным здесь является проверка наличия кода ENTER, поэтому этот параметр проверяется в первую очередь. В случае обнаружения этого кода осуществляется переход на подпрограмму 13TH CONTROL, которая увеличивает содержимое регистра BC на 4 для того, чтобы анализу не подверглись номер и длина строки Бейсика. Параллельно с увеличением на 4 регистра BC, происходит уменьшение регистра HL, который служит счетчиком, и по которому определяется окончание работы программы.

Если же содержимое ячейки не является кодом ENTER, то осуществляется проверка его на код INK CONTROL и PAPER CONTROL. В случае, если оно соответствует какому-либо из этих значений, то осуществляется перевод на подпрограмму обработки INC_CONTROL или PAP_CONTROL соответственно.

Эти подпрограммы осуществляют принудительный ввод кодов черного и белого цвета, аналогично тому, как это осуществила Бейсик-программа.

Следующей идет проверка наличия кода BACKSPACE. Соответствующая подпрограмма осуществляет замену этого кода на код пробела 32, поскольку в большинстве случаев BACKSPACE используется для скрытия каких-то определенных элементов программы, которые, однако оказываются доступны просмотру при замене его на код SPACE. Фактически, если бы мы ограничились только блоком ANALIZ, то мы имели бы полный аналог описанной выше программы на Бейсике. Однако, поскольку набор управляющих кодов не ограничивается лишь только INK CONTR, PAPER CONTROL и

BACKSPACE, а существуют еще OVER CONTROL, BRIGHT CONTROL, INVERSE CONTROL, FLASH CONTROL, AT и TAB CONTROL (а для защиты от листинга может с успехом использоваться практически любой из них), то программа в машинных кодах имеет расширение, осуществляющее контроль наличия всех вышеописанных управляющих кодов. В случае их обнаружения осуществляется замена их кодом пробела - 32 с использованием подпрограммы BACK. Блок NEXT PARAMETRS изменяет содержимое контрольных регистров HL и BC таким образом, чтобы осуществлялся анализ следующих ячеек памяти.

Блок END CONTROL осуществляет контроль окончания программы по содержимому регистра HL. Если счетчик HL содержит 0, то осуществляется возврат в вызывающую программу по команде SET. Контроль осуществляется следующим образом.

Сначала проверяется содержимое старшего разряда регистра HL и сравнивается с 0. Если оно равно 0, то осуществляется проверка содержимого младшего разряда данного регистра. Если и оно равно 0, то осуществляется выход и программа продолжает анализ содержимого текущих ячеек оперативной памяти компьютера.

Мы с Вами рассмотрели принцип работы программы блокировки управляющих кодов. Теперь рассмотрим некоторые аспекты ее практического применения.

Как уже было отмечено выше, данная программа может работать в любом свободном месте оперативной памяти. Это достигается за счет использования команд относительного перехода JR (ввиду того, что объем программы незначителен, применение функции JR вполне допустимо).

Как было подчеркнуто, данная программа допускает совместное использование с программой автозапуска. Это допустимо потому, что программа блокировки автозапуска формируется с адреса 62000 и занимает 25 байтов. С учетом этого, можно записать общий блок кодов как самостоятельную программную единицу, подав команду:

```
SAVE "BLOKIR" CODE 62000, 135
```

Теперь вы можете, загрузив этот блок, вызывать данные процедуры, давая команды:

```
RANDOMIZE USR 62000
```

для процедуры блокировки автозапуска

и

```
RANDOMIZE USR 62030
```

для блокировки управляющих кодов.

Примечание. Необходимо отметить, что перед тем, как выгрузить данный блок кодов на магнитофон, необходимо сформировать программу блокировки автозапуска, начиная с адреса 62000, что достигается путем использования программы на Бейсике, описанной в 2.2.3. Для того, чтобы сформировать процедуру обработки управляющих кодов можно тоже воспользоваться Бейсиком, используя десятичную последовательность кодов как блок DATA.

Для этого можно использовать достаточно простую программу.

```
10 FOR I=63030 TO 62130
20 READ N: POKE I, N
30 NEXT I
40 DATA 33,200,0,237,75,83,92,205
```

и т.д. в соответствии с приведенными ниже значениями:

```
Programming by Mihailenko Vadim. All rights reserved. Mensk1991.
Mihailenko Vadim driver system for "EDITAS-48" files.
Special for "INFORCOM" Corporation.

33,   200,  0,    237,  75,   83,   92,   205,
84,   31,   48,   85,
10,   254,  13,   40,
51,   254,  16,   40,   57,   254,  16,   40,
60,   254,  8,    40,
63,   254,  18,   40,
59,   254,  19,   40,
```

```
55,    254,  20,   40,
51,    254,  21,   40,
47,    254,  22,   40,
43,    254,  23,   40,
39,    43,   3,    124,
254,   0,    40,   2,
24,    201,  125,  254,
0,     200,  24,   195,
43,    43,   43,   43
3,     3,    3,    3,
24,    185,  43,   3,
62,    0,    2,    24,
178,   43,   3,    62,
7,     2,    24,   171,
62,    32,   2,    24,
166,   207,  20,   0,
```

Если же вы желаете сформировать данную последовательность кодов в другом месте оперативной памяти, то необходимо произвести соответствующие изменения в строке 10 Бейсик-программы, указав вместо 62030 необходимое значение.

Теперь рассмотрим небольшую особенность данной программы, связанную с длиной обрабатываемой Бейсик-программы, т.е. с длиной области оперативной памяти, которая очищается от защитных управляющих кодов. В моем варианте мы обрабатываем 200 байтов оперативной памяти, однако бывают случаи, когда этого оказывается недостаточно. Чтобы увеличить обрабатываемую область, необходимо увеличить число, заносимое в регистр HL. Если Вы сформировали данную программу с адреса 62030, то значение, заносимое в HL будет характеризоваться двумя байтами 62031 и 62032, причем сначала идет младший байт, а потом старший. Если Вы хотите сделать величину обрабатываемой области X, то Вам необходимо ввести соответствующую строку с клавиатуры:

```
LET A=INT(X/255): POKE 62032,A; LET B=X-A*255: POKE 62031,B
```

Если же Вы хотите подбирать значение переменной X, то можно оформить эту последовательность операндов, как строку Бейсик-программы.

Следует отметить еще одну немаловажную деталь, характеризующую работу данной программы. Ввиду того, что достаточно часто для защиты используются встроенные процедуры в машинных кодах (они непроизвольно останавливают листинг с выдачей сообщения INVALID COLOR или NUMBER TOO BIG) совместно с методом зануления, а данная программа блокировки управляющих кодов не делает различие между строками Бейсика и встроенными процедурами, то происходит их полная переработка (анализ), что, с одной стороны, позволяет Вам получить полный листинг программы, но с другой стороны, ввиду изменения во встроенных процедурах машинных кодов мы не можем вызывать эти процедуры сразу после просмотра листинга, чтобы увидеть весь эффект их действия. Для того, чтобы все же наблюдать данный эффект, необходимо вновь загрузить исходный Бейсик-файл и осуществлять проверку до использования программы блокировки управляющих кодов.

Примечание ИНФОРКОМа. Мы не сможем запускать программу со снятыми кодами и по причине отмеченной выше - в связи с тем, что коррумпируются числа в БЕЙСИК-строках.

Внимание!

Данная машиннокодовая программа для блокировки действия управляющих кодов написана Михайленко Вадимом. При использовании в разработках указывать автора.

(Продолжение следует)

Дж. Хардман, Э. Хьюзон
Перевод Тихоновой Н.А.

40 ЛУЧШИХ ПРОЦЕДУР

Окончание.
Начало см. с. 17-28, 61-70, 105-110.

8.5 Составление списка переменных.

Длина: 94
Количество переменных: 0
Контрольная сумма: 10295
Назначение:

Эта подпрограмма составляет список имен всех переменных, имевшихся в настоящий момент в памяти.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если переменных в памяти нет, программа возвращается в BASIC.

Комментарий:

Это большая помощь при отладке программ, особенно длинных и сложных.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
           RES 0,(IY+2)       253   203   2      134
           LD HL,(23627)      42    75    92
N_VAR      LD A,              13    62    13
           RST 16             215
           LD A,              32    62    32
           RST 16             215
           LD A,(HL)          126
           CP 128             254   128
           RET Z              200
           BIT 7,A            203   127
           JR Z,BIT_5         40    62
           BIT 6,A            203   119
           JR Z,N_BIT         40    31
           BIT 5,A            203   111
           JR Z,STR_AR        40    9
           SUB 128            214   128
           LD DE,19           17    19    0
PRINT      RST 16             215
           ADD HL,DE          25
           JR N_VAR           24    225
STR_AR     SUB 96             214   96
           RST 16             215
           LD A, 36           62    36
BRACK      RST 16             215
           LD A,40            62    4     0
           RST 16             215
           LD A,41            62    41
POINT      INC HL             35
           LD E,(HL)          94
           INC HL             35
           LD D,(HL)          86
           INC HL             35
           JR PRINT           24    234
```

```
N_BIT      BIT 5,A          203   111
           JR Z,ARRAY       40    19
           SUB 64           214   64
           RST 16           215
NEXT_C     INC HL           35
           LD A,(HL)        126
           BIT 7,A          203   127
           JR NZ,LAST_C     32    3
           RST 16           215
           JR NEXT_C        24    247
LAST_C     SUB 128          214   128
JUMP       LD DE,6          17    6     0
           JR PRINT         24    211
ARRAY      SUB 32           214   32
           JR BRACK         24    216
BIT_5      BIT 5,A          203   111
           JR NZ,JUMP       32    243
           ADD A,32         198   32
           RST 16           215
           LD A, 36         62    36
           JR POINT         24    211
```

Как она работает:

Бит 0 байта по адресу 23612 сбрасывается, чтобы символы, выводимые на печать, появлялись в верхней части экрана. В HL загружается адрес области переменных. В аккумулятор загружается признак ENTER и вызывается подпрограмма ПЗУ по адресу 16. В аккумулятор затем загружается код пробела, и вновь вызывается та же самая подпрограмма ПЗУ.

В аккумулятор загружается байт по адресу в HL. Если значение этого байта равно 128, программа возвращается в BASIC, т.к. достигнут конец области переменных.

Если бит 7 аккумулятора установлен в 0, программа переходит к 'BIT_5', т.к. встретились строковая переменная или число, имя которых состоит только из одной буквы. Проверяется 6 бит аккумулятора. Если он равен 0, делается переход к 'N_BIT', т.к. определены массив или число, имя которых более, чем одна буква. Если бит 5 аккумулятора равен 0, программа переходит к 'STR_AR'

Программа достигает этой точки, если найденная переменная является управляющей переменной цикла FOR/NEXT. В этом случае из аккумулятора вычитается 128 и результатом является код символа для вывода на печать. В пару DE загружается число 19, указывая на следующую переменную при прибавлении к HL. Символ в аккумуляторе выводится на печать, DE прибавляется к HL, а программа возвращается к поиску следующей переменной 'N_VAR'.

Если программа находит строковый массив (достигает 'STR_AR'), то из аккумулятора вычитается число 96, что дает код имени найденного массива. Это значение выводится на печать, используя подпрограмму ПЗУ. Знак доллара и левая скобка выводятся на печать, а в аккумулятор загружается код правой скобки. HL увеличивается, указывая на байты, содержащие длину массива. Это значение загружается в DE, так что прибавление к HL дает адрес следующей переменной. Делается переход к 'PRINT', где правая скобка выводятся на печать, и DE прибавляется к HL.

В процедуре 'N_BIT' проверяется бит 5 аккумулятора. Если он установлен в 0, т.е. это числовой массив, то происходит переход к 'ARRAY'. Если он установлен в 1, то это числовая переменная, имя которой длиннее, чем одна литера. Из аккумулятора вычитается 64, а полученный в результате символ выводится на печать. Затем программа выполняет цикл, выводя на печать каждый встретившийся символ, до тех пор, пока находится хоть один символ с битом 7, установленным в 1. Из кода этого последнего символа вычитается 128, в DE загружается смещение для следующей переменной, а программа переходит к 'PRINT'.

Если массив найден, 32 вычитается из аккумулятора, чтобы получить правильный код, и делается переход на поиск скобок ' BRACK'.

В процедуре 'BIT_5', если найдено число, имя которого имеет только одну букву,

программа возвращается к 'JUMP'.

Окончание подпрограммы работает, когда встречающиеся переменные - строковые. Прибавление 32 к аккумулятору дает код для вывода на печать. Наконец, в аккумулятор загружается код знака доллара и делается переход к 'POINT'

8.6 Поиск строки.

Длина: 155

Количество переменных: 2

Контрольная сумма: 17221

Назначение:

Эта программа осуществляет поиск по БЕЙСИК программе и выводит каждую строку, содержащую набор символов, определенных пользователем.

Переменные:

Имя - data start

Длина - 2

Адрес 23296

Комментарий: адрес первого байта данных.

Имя - string length

Длина - 1

Адрес - 23298

Комментарий: число символов в строке.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если в памяти нет БЕЙСИК-программы или символьная строка имеет нулевую длину - возврат в БЕЙСИК.

Комментарии:

Время выполнения этой программы пропорционально двум величинам: длине строковой переменной и длине БЕЙСИК программы. Строка для поиска должна быть помещена в ячейку выше RAMTOP, а адрес первого байта строки должен быть помещен в ячейки 23296/7. Длина строки должна быть сохранена в ячейке 23298.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР         ЧИСЛА ДЛЯ ВВОДА
            RES 0,(IY+2)      253   203   2     134
            LD IX,(23296)     221   42    0     91
            LD HL,(23635)     42    83    92
RESTAR      LD A,(23298)      58    2     91
            LD E,A            95
            CP 0              254   0
            RET Z             200
            PUSH HL           229
RESTOR      PUSH IX           221   229
            POP BC            193
            LD D,             0     22    0
            INC HL            35
            INC HL            35
            INC HL            35
CHECK       INC HL            35
            PUSH DE           213
            LD DE,(23627)     237   91    75    92
            AND A             167
            SBC HL, DE        237   82
            ADD HL, DE        25
```

```
        POP DE          209
        JR C,ENTER      56    4
        POP HL          225
        RET             201
LONG_J  JR RESTAR       24    223
ENTER   LD A, (HL)      126
        CP 13           254   13
        JR NZ,NUMBER    32    5
        INC HL          35
        POP BC          193
        PUSH HL         229
        JR RESTOR       24    221
NUMBER  CALL 6326       205   182   24
        JRNZ COMPAR     32    8
        DEC HL          43
DIFFER  PUSH IX         221   229
        POP BC          193
        LD D,           0     22    0
        JR CHECK        24    216
COMPAR  LD A,(BC)       10
        CP (HL)         190
        JR NZ,DIFFER    32    245
        INC BC          3
        INC D           20
        LD A, D         122
        CP E            1     87
        JR NZ,CHECK     32    206
        LD A,           13    62    1     3
        RST 16          215
        POP HL          225
        PUSH HL         229
        LD B,(HL)       70
        INC HL          35
        LD L,(HL)       110
        LD H,B          96
        LD DE,1000      17    232   3
        LD A,47         62    47
THOUS   INC A           60
        AND A           167
        SBC HL,DE       237   82
        JR NC,THOUS     48    250
        ADD HL,DE       25
        RST 16          215
        LD DE,100       17    100   0
        LD A,47         62    47
HUNDR   INC A           60
        AND A           157
        SBC HL,DE       237   82
        JR NC,HUNDR     48    250
        ADD HL,DE       25
        RST 16          215
        LD DE,10        17    10    0
        LD A,           47    62    47
TENS    INC A           60
        AND A           167
        SBC HL,DE       237   82
        JR NC,TENS      48    250
        ADD HL,DE       25
        RST 16          215
        LD A,L          125
        ADD A,48        198   48
        RST 16          215
        POP HL          225
        INC HL          35
        INC HL          35
```

```
          INC HL           35
NEXT_C    INC HL           35
          LD A,(HL)        126
LINEND    CP 13            254   13
          JR NZ,CHR_14     32    4
          RST 16           215
          INC HL           35
          JR LONG_J        24    155
CHR_14    CALL 6326        205   182   24
          JR Z,LINEND      40    243
          CP 32            254   32
          JR C,NEXT_C      56    237
          RST 16           215
          JR NEXT_C        24    234
```

Как она работает:

Бит 0 байта, хранящегося по адресу 23612 сбрасывается, чтобы символы, выводимые на печать, появлялись в верней части экрана. В IX загружается адрес первого байта данных. Это позволяет загрузить этот адрес в другую пару регистров, используя в меньшей степени буфер принтера. В HL, загружается адрес начала БЕЙСИК-программы.

В аккумулятор загружается длина эталонной строки, и это значение копируется в E-регистр. Если длина строки равна 0, программа возвращается в БЕЙСИК. Адрес в HL помещается в стек, храня положение искомой в настоящий момент строки в памяти.

Адрес данных копируется из IX в BC для большей доступности. В D-регистр загружается 0, т. е. количество найденных символов, равнозначных введенным данным. Пара регистров HL увеличивается на 3, указывая на старший байт указателя длины строки. HL увеличивается, указывая на следующий символ. Пара регистров DE сохраняется в стеке.

В DE загружается адрес области переменных, и это значение вычитается из HL. Если результат отрицательный, программа переходит к 'ENTER' после восстановления HL и возвращения из стека DE. Если результат был положительным, стек восстанавливается до своего первоначального размера и выполняется возврат в БЕЙСИК, т. к. достигнут конец БЕЙСИК-программы.

В процедуре 'ENTER' в аккумулятор загружается байт, хранящийся по адресу в HL. Если это не признак ENTER, происходит переход к 'NUMBER'. Если признак ENTER найден, HL увеличивается, указывая на начало следующей строки. Адрес предыдущей строки удаляется из стека и замещается новым значением в HL. Затем делается переход к 'RESTOR'. В процедуре 'NUMBER' вызывается подпрограмма ПЗУ, расположенная там по адресу 6326. Если символ в аккумуляторе является признаком NUMBER (CHR_14), HL увеличивается, указывая на первый символ после пятибайтного представления числа, определенного подпрограммой ПЗУ. Если признак NUMBER не обнаружен, программа переходит к 'COMPAR', иначе HL уменьшается и программа переводит к 'DIFFER'. BC копируется из IX, количество найденных символов сбрасывается в 0 и делается переход к 'CHECK' .

В процедуре 'COMPAR' в аккумулятор загружается байт, хранящийся по адресу в BC. Если это значение не то же самое, что и байт, хранящийся по адресу в HL, программа возвращается к 'DIFFER'.

BC увеличивается, указывая на следующий байт данных, и количество определенных символов увеличивается. Если это значение не равно длине символьной строки, программа возвращается к 'CHECK'. В аккумулятор загружается код признака ENTER, и это значение выводится на печать, используя команду RST 16. Адрес строки для вывода на печать загружается из стека в HL, номер строки затем копируется в HL через B-регистр. В DE загружается 1000 и в аккумулятор загружается значение, на 1 меньшее, чем код символа "0". Аккумулятор уменьшается, а DE повторно вычитается из HL до тех пор, пока HL не станет отрицательным. Затем DE прибавляется к HL, чтобы получить положительный остаток. Символ из аккумулятора выводится на печать.

Вышеописанный прием повторяется затем для DE=100 и DE=10. Затем остаток загружается в аккумулятор, прибавляется 48, и в результате полученный символ выводится

на печать.

Адрес начала строки восстанавливается из стека и загружается в HL. Затем HL увеличивается, указывая на старший байт указателя длины строки, HL увеличивается, и байт с адресом в HL загружается в аккумулятор. Если этот байт не является признаком ENTER, делается переход к 'CHR_14', иначе ENTER выводится на печать, HL увеличивается, и программа возвращается к 'RESTAR'.

В процедуре 'CHR_14' вызывается подпрограмма ПЗУ по адресу 6326. Если символ в аккумуляторе является признаком числа, HL увеличивается, указывая на первый символ, стоящий после найденного числа. Этот символ загружается в аккумулятор и делается переход к 'LINEND'. Затем, если символ в аккумуляторе имеет код меньший, чем 32, подпрограмма возвращается к 'NEXT_C'. Если код больше, чем 31, найденный символ выводится на печать и происходит переход к 'NEXT_C'.

8.7 Поиск и замещение строки.

Длина: 85

Количество переменных: 3

Контрольная сумма: 8518

Назначение:

Программа шлет символьную строку в БЕЙСИК-программе и делает замену каждой найденной строки стринга на другую строку такой же длины.

Переменные:

Имя - old data start

Длина - 2

Адрес - 23296

Комментарий: адрес первого байта замещаемой строки.

Имя - string length

длина - 1

Адрес - 23298

Комментарий: длина замещаемой строки.

Имя - new data start

Длина - 2

Адрес - 23299

Комментарий: адрес первого байта замещающей строки.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если длина строки равна 0 или БЕЙСИК-программы в памяти нет, то процедура возвращается непосредственно в БЕЙСИК.

Комментарий:

Время выполнения этой программы зависит от длины строки и от длины БЕЙСИК-программы.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР         ЧИСЛА ДЛЯ ВВОДА
           LD IX,(23296)     221   42    0     91
           LD HL,(23635)     42    83    92
           LD A,(23298)      58    2     91
           LD E,A            95
           CP 0              254   0
           RET Z             200
           DEC HL            43
NEWLIN     INC HL            35
```

```
        INC HL          35
        INC HL          35
        INC HL          35
        JR RESET        24   23
CHECK   INC HL          35
        PUSH DE         213
        LD DE,(23627)   237  91   75   92
        AND A           167
        SBC HL,DE       237  82
        ADD HL,DE       25
        POP DE          209
        RET NC          208
        LD A,(HL)       126
        CP 13           254  13
        JR Z,NEWLIN     40   233
        CALL 6326       205  182  24
        JR NZ, COMPAR   32   8
        DEC HL          43
RESET   PUSH IX         221  229
        POP BC          193
        LD D,0          22   0
        JR CHECK        24   226
COMPAR  LD A,(BC)       10
        CP (HL)         190
        JR NZ,RESET     32   245
        INC BC          3
        INC D           20
        LD A,D          122
        CP E            187
        JR NZ,CHECK     32   216
        PUSH HL         229
        LD D,0          22   0
        AND A           167
        SBC HL,DE       237  82
        LD D, E         83
        LD BC,(23299)   237  75   3    91
        INC D           20
NEXT_CH INC HL          35
        DEC D           21
        JR Z, FINISH    40   5
        LD A,(BC)       10
        LD (HL),A       119
        INC BC          3
        JR NEXT_C       24   247
FINISH  POP HL          225
        JR RESET        24   215
```

Как она работает:

В IX загружается адрес замещающей строки. Это значение должно быть выше RAMTOP. В HL загружается адрес начала программной области, а в аккумулятор загружается длина строки, которая копируется в E-регистр для дальнейшего использования в программе. Если длина строки равна 0, программа возвращается в БЕЙСИК.

Устанавливается HL, указывая на старший байт следующего указателя БЕЙСИК-строки и делается переход к 'RESET'.

В процедуре 'CHECK' HL увеличивается, указывая на следующий символ. DE сохраняется в стеке и загружается адресом области переменных. Если HL не меньше, чем DE, конец программы достигнут, и после восстановления DE из стека программа возвращается в БЕЙСИК.

В аккумулятор загружается символ по адресу в HL, Если это значение является знаком ENTER, программа возвращается к 'NEWLIN'. Если аккумулятор не содержит знак NUMBER (символ 14), делается переход к 'COMPAR', иначе HL увеличивается на 5, так что HL указывает на пятый байт найденного числа.

В процедуре 'RESET' в BC загружается адрес строки для поиска. Регистр D устанавливается в 0 для хранения количества символов в строке, найденной к тому времени. Программа затем возвращается к 'CHECK'.

В процедуре 'COMPARE' в аккумулятор загружается символ строки, на который указывает пара регистров BC. Если это значение отличается от байта по адресу в HL, программа переходит к 'RESET'. BC увеличивается, указывая на следующий символ в строке, регистр D, счетчик, увеличивается. Если это значение не равно длине строки, программа возвращается к 'CHECK'.

Если строка найдена, HL сохраняется на стеке, так что программа начинает поиск для следующего случая с этого адреса. В DE загружается длина строки и это значение вычитается из HL, давая значение на единицу меньше, чем стартовый адрес. Длина строки затем загружается в D для использования ее в качестве счетчика. В BC загружается стартовый адрес замещающей строки, а регистр D увеличивается. Регистр HL увеличивается, указывая на следующую ячейку, а счетчик уменьшается.

Если счетчик равен 0, HL восстанавливается из стека и делается переход к 'RESET' для следующего случая. В аккумулятор загружается символ, на который указывает BC и это значение помещается в ячейку по адресу в HL. BC увеличивается, указывая на следующий символ, а программа возвращается к 'NEXT CHAR'.

8.8 Поиск подстроки.

Длина: 168

Количество переменных: 0

Контрольная сумма: 19875

Назначение:

Эта программа возвращает позицию подстроки (B$) в главной строке (A$) или 0 в случае ошибки.

Вызов программы:

```
LET P = USR адрес контроль ошибок:
```

Если строка не существует или если длина подстроки равна нулю или если длина подстроки больше, чем длина главной строки, программа возвращает значение 0. Если ошибки нет, но подстрока не найдена в главной строке, программа также дает 0.

Комментарий:

После выхода из программы в машинных кодах переменная P (может быть использована любая другая переменная) будет хранить искомое значение. Строки, к которым делается обращение, не могут быть массивами данных. Для изменения используемых строк числа, отмеченные звездочкой, должны быть изменены. 66* - это подстрока, 65* - главная строка. Для изменения эти числа необходимо заменить кодами требуемых символов. Например, если Вы хотите найти позицию H$ в G$, то соответственно надо будет ввести 71 (код G) и 72 (код H)

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА      АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
           SUB A              151
           LD B, A            71
           LD B, A            79
           LD D, A            87
           LD E, A            95
           LD HL,(23627)      42    75    92
NEXT_V     LD A,(HL)          126
           CP 128             254   128
           JR Z,NOT_FD        40    95
           BIT 7, A           203   127
           JR NZ,FOR_NX       32    41
           CP 96              254   96
           JR NC,NUMBER       48    29
```

```
                CP 65              254   65*
                JR NZ,SUBSTR       32    2
                LD D, H            84
                LD E, L            93
SUBSTR          CP 66              254   66*
                JR NZ,CHECK        32    2
                LD B,H             68
                LD C,L             77
CHECK           LD A,D             122
                OR E               179
                JR Z,STRING        40    4
                LD A,B             120
                OR C               177
                JR NZ,ROUND        32    38
STRING          PUSH DE            213
                INC HL             35
                LD E,(HL)          94
                INC HL             35
                LD D,(HL)          86
ADD             ADD HL,DE          25
                POP DE             209
                JR INCRS           24    5
NUMBER          INC HL             35
                INC HL             35
                INC HL             35
                INC HL             35
                INC HL             35
INCRS           INC HL             35
                JR NEXT_V          24    206
FOR_NX          CP 224             254   224
                JR C, N_BIT        56    6
                PUSH DE            213
                LD DE,18           17    18    0
                JR ADD             24    234
N_BIT           BIT 5,A            203   111
                JR Z,STRING        40    225
NEXT_B          INC HL             35
                BIT 7,(HL)         203   126
                JR Z,NEXT_B        40    251
                JR NUMBER          24    227
FOUND           EX DE,HL           235
                INC HL             35
                INC HL             35
                PUSH HL            289
                PUSH HL            229
                INC BC             3
                PUSH BC            197
                LD A,(BC)          10
                LD E,A             95
                INC BC             3
                LD A,(BC)          10
                LD D,A             87
                OR E               179
                JR Z,ZERO          40    11
                PUSH DE            213
                LD A,(HL)          126
                DEC HL             43
                LD L,(HL)          110
                LD H,A             103
                AND A              167
                SBC HL,DE          237   82
                JR NC,CONTIN       48    8
                POP BC             193
ZERO            POP BC             193
                POP BC             193
```

```
ERROR       POP BC           193
NOT_FD      LD BC,0          10     0
            RET              201
CONTIN      POP IX           221    285
            POP BC           193
            EX DE,HL         235
            POP HL           225
            INC BC           3
            INC BC           3
SAVE        INC HL           35
            PUSH HL          229
            PUSH BC          197
            PUSH IX          221    229
            PUSH DE          213
COMPAR      LD A,(BC)        10
            CP (HL)          190
            JR Z,MATCH       40     12
            POP DE           209
            POP IX           221    225
            POP BC           193
            POP HL           225
            LD A,D           122
            OR E             179
            JR Z,ERROR       40     225
            DEC DE           27
            JR SAVE          24     234
MATCH       INC HL           35
            INC BC           3
            PUSH HL          229
            DEC IX           221    43
            PUSH IX          221    229
            POP HL           225
            LD A,H           124
            OR L             181
            POP HL           225
            JR NZ,COMPAR     32     227
            POP DE           209
            POP DE           209
            AND A            167
            SBC HL,DE        237    82
            POP DE           209
            POP DE           209
            POP DE           209
            AND A            167
            SBC HL, DE       237    82
            LD B,H           68
            LD C,L           77
            RET              201
```

Как она работает:

В аккумулятор, пару регистров ВС и пару регистров DE загружается 0. Позднее в программе в ВС будет установлен адрес B$, а в DE будет установлен адрес A$. В HL загружается адрес начала области программных переменных.

В аккумулятор загружается байт из адреса, находящегося в HL. Если аккумулятор содержит число 128 программа переходит к 'NOT_FD', т.к. достигнут конец области программных переменных. Если бит 7 аккумулятора установлен в 1, делается переход к 'FOR_NX', так как найденная переменная - не строковая и не число, имя которого состоит только из одной литеры. Если аккумулятор содержит число большее, чем 95, делается переход к 'NUMBER'.

Для достижения этого этапа строка должна быть найдена. Если в аккумуляторе содержится число 65, определяется местонахождение строки A$, а содержимое HL копируется в DE. Если аккумулятор содержит число 66, определяется строка B$, а HL

копируется в BC. Если DE не равно 0, и BC не равно 0, определяется местонахождение обеих строк, и программа переходит к 'FOUND'.

Если программа достигает процедуры 'STRING', DE сохраняется в стеке и загружается длиной найденной строки. Это значение прибавляется к адресу старшего байта указателей строки и сохраняется в HL.. DE восстанавливается из стека и делается переход к 'INCRS'.

В процедуре 'NUMBER' HL увеличивается в пять раз, указывая на последний байт найденного числа. HL затем увеличивается, указывая на следующую переменную, и происходит переход к 'NEXT_V'.

В процедуре 'FOR_NX', если аккумулятор содержит число меньше, чем 224, делается переход к 'N_BIT', т.к. встретившаяся переменная не является управляющей переменной цикла FOR-NEXT. Если значение аккумулятора больше, чем 223, то число 18 прибавляется к HL, указывая на последний байт переменной цикла и программа возвращается к 'INCRS'.

Если программа достигает 'N_BIT', и бит 5 аккумулятора установлен в 0, делается переход к 'STRING', чтобы загрузить в HL адрес следующей переменной, т.к. найден массив.

Если программа достигает 'NEXT_B', найдено число, имя которого больше одного символа по длине. Т.о., HL увеличивается до тех пор, пока не укажет на последний символ имени переменной, а затем делается переход к 'NUMBER'.

В процедуре 'FOUND' в HL загружается адрес строки A$, и это значение увеличивается дважды, чтобы получить адрес старшего байта указателей. Это значение затем сохраняется в стеке дважды. BC увеличивается, указывая на младший байт указателей подстроки B$. Адрес в BC затеи сохраняется в стеке, в DE загружается длина строки B$, и, если это значение равно 0, делается переход к 'ZERO'. Затем DE помещается в стек. В HL загружается длина строки a$, и, если это значение не меньше, чем DE, программа переходит к 'CONTIN'. Указатель стека затем восстанавливается, в BC загружается 0, и программа возвращается в БЕЙСИК.

В процедуре 'CONTIN' в IX устанавливается длина строки B$, а в BC помещается адрес младшего байта указателей для подстроки B$. В DE загружается разность длин строк A$ и B$, а в HL загружается адрес старшего байта указателей для A$. BC затем увеличивается дважды, чтобы получить адрес первого символа в подстроке B$. HL увеличивается, указывая на следующий символ строки A$. HL, BC, IX и DE затем сохраняются на стеке. В аккумулятор загружается байт по адресу в BC, и, если это значение равно значению байта по адресу в HL, делается переход к 'MATCH'. DE, IX, BC и HL затем восстанавливаются из стека. Если DE содержит 0, делается переход к 'ERROR', т.к. подстроки B$ нет в строке A$. Счетчик DE уменьшается, и программа возвращается к 'SAVE'.

Если программа достигает процедуры 'MATCH', HL и BC увеличиваются, указывая на следующий символ A$ и B$ соответственно. HL затем сохраняется в стеке. IX, счетчик, уменьшается и после восстановления HL из стека, если IX не содержит 0, программа возвращается к 'COMPAR'.

Для достижения этого этапа местонахождение подстроки B$ в строке A$ уже должно быть определено. Длина подстроки B$ вычитается из HL, а затем адрес старшего байта указателей для строки A$ вычитается из HL. Результат - это позиция подстроки B$ в строке A$. Это значение копируется в пару регистров BC, и программа возвращается в БЕЙСИК.

* * *

Заканчивая печать книги Дж. Хардмана и Э. Хьюзона "40 лучших процедур", нам хотелось бы дать небольшой комментарий, который касается формата программных переменных в "Спектруме". Дело в том, что процедуры, представленные в этом последнем заключительном блоке широко оперируют с ними. Те, кто не имеют фирменную инструкцию по "Спектруму" (книга Виккерса), могут быть с этим форматом и не знакомы, а мы в своих работах до сих пор как-то к этому вопросу не обращались.

Те, кому этот вопрос интересен, могут прочитать комментарий на стр. 44.

Формат данных в "Спектруме"

Комментарий к стр. 43

Данные в "Спектруме" хранятся в виде переменных и массивов в специально выделенной для этого области памяти. Эта область начинается непосредственно за областью, в которой размешается текст БЕЙСИК-программы.

На начало области программных переменных указывает двухбайтная системная переменная VARS. Она расположена по адресу 23267 (5AE3H).

Конец области программных переменных задан специальным маркером - это байт, значение которого равно 80Н (128),

"Спектрум" различает несколько разных типов переменных. Это:

- обычная числовая переменная, имя которой состоит из одной буквы, например x;
- числовая переменная, имя которой состоит из более, чем одной буквы, например row;
- числовой массив, например a(5) или b (3,3,40);
- переменные, управляющие циклами FOR. .. NEXT, например i;
- строковые переменные, например a$;
- строковые массивы, например b$(10,40);

Числовая переменная с именем из одной буквы.

Занимает 6 байтов. В первом байте хранится ее имя. В последующих пяти - ее значение в интегральной форме. Об интегральном представлений действительных чисел см. "Первые шаги в машинном коде". Первый байт имеет следующую раскладку:

0	1	1	б	у	к	в	а

На то, что это простая переменная указывает специфическое расположение первых трех битов.

Числовая переменная с именем более чем из одной буквы.

Ее первый байт имеет следующий формат:

1	0	1	б	у	к	в	а

Прочие байты имени (кроме последнего) имеют следующий формат:

0	б	у	к	в	а	.	.

Последний байт имени:

1	б	у	к	в	а	.	.

За именем следуют 5 байтов для выражения самого числа в интегральной форме.

Числовой массив.

Первый байт:

1	0	0	б	у	к	в	а

Байты 2, 3 содержат полную длину всех элементов (по 5 байтов на каждый элемент массива) плюс по 2 байта на каждую размерность массива плюс один байт на указание количества размерностей, т. е. здесь содержится указание на конец массива.

Байт 4 содержит размерность массива.

Байты 5,6 содержат количество элементов в первой измерении.

Если размерность массива более чем 1, то:

Байты 7, 8 содержат количество элементов во втором измерении;
и т. д.

После этого идут сами элементы массива по пять байтов на каждый элемент. Для многомерных массивов порядок следования данных следующий: b(1,1), b(1,2), b(1,3), b(2,1).....b(3,3)

Переменные цикла.

Первый байт:

1	1	1	б	у	к	в	а

Далее:
5 байтов - текущее значение;
5 байтов - конечное значение;
5 байтов - шаг;
2 байта - номер строки возврата;
1 байт - номер оператора в строке, к которому выполняется возврат.

Символьная переменная.

Первый байт:

0	1	0	б	у	к	в	а

Далее:
2 байта - длина строки.
x байтов - текст строки.

Символьный массив.

Первый байт:

1	1	0	б	у	к	в	а

Далее:
2 байта - указание на конец массива;
1 байт - размерность;
2 байта - длина в первом измерении:
2 байта - длина в последнем измерении;
Далее:
по одному байту на каждый элемент.

ЭЛЕМЕНТАРНАЯ ГРАФИКА В МАШИННЫХ КОДАХ

(Глава из книги "Элементарная графика")

Сегодня мы представляем Вам наше новое издание. Оно посвящено графике "Спектрума" и будет выпущено в четырех томах.

Первый том называется "Персональный компьютер ZX-СПЕКТРУМ. Элементарная графика".

Вашему вниманию предлагается одна глава из этой книги.

Книга готова к изданию и мы ждем заявок и предложении от организаций, способных приняться за ее издание и распространие на взаимовыгодных условиях.

ЭЛЕМЕНТАРНАЯ ГРАФИКА В МАШИННЫХ КОДАХ

Когда мы слышим сочетание слов "компьютерная графика", то нам сразу представляются сложные многоцветные трехмерные изображения и желательно, чтобы при этом что-то двигалось, лучше, если побольше, побыстрее и как можно более плавно.

Все это, конечно же так, но начинается компьютерная графика, тем не менее, не с этого. Когда Вы даете команду компьютеру PRINT "*" и он это делает, Вы уже работаете с графикой, хотя об этом и не задумывались. Можно считать так, что как только Вы делаете что-то, что приводит к изменению изображения на экране Вашего телевизора или монитора, Вы уже занимаетесь компьютерной графикой, особенно если Вам понятно, почему эти изменения происходят именно так, а не иначе и в какой-то степени можете ими управлять.

Итак, если Вы из БЕЙСИКа напечатаете звездочку на Вашем экране, то вам потребуется изрядная доля воображения для того, чтобы считать, что это компьютерная графика и убедить своих друзей, что у Вас есть дизайнерские способности. А что, если вы сделаете то же самое из машинного кода? А если при этом Вы ее не напечатаете, а нарисуете по точкам? Все дело принимает совсем другой оборот, не правда ли? Итак, все дело не в терминах, а в целенаправленности Ваших усилий, в способности задумать что-то и найти способы, как это реализовать.

Если вы не нашли до сих пор достаточно времени, чтобы освоить программирование в машинных кодах, то не только сузили круг своих технических возможностей, но и ограничили возможности для самовыражения и для дальнейших творческих поисков. Наш пример с печатанием звездочки здесь как раз и служит для того, чтобы дать представление о том, что и в графике значение имеет не только конечный результат, но и путь, который к нему привел.

В этой главе мы попробуем дать Вам те основы, которые необходимы дли того, чтобы начать эксперименты с графикой из машинного кода. Как и в любом другом вопросе, связанном с программированием на "Спектруме", мы не надеемся дать полную и исчерпывающую картину. Как и всегда, "ИНФОРКОМ" видит главную задачу в том, чтобы помочь сделать первый шаг, а дальше Вы сами раскроете свои таланты.

Первое, что нам потребуется - это вспомнить концепцию потоков и каналов "Спектрума". Те, кто более глубоко заинтересуются этой концепцией, могут найти информацию в "ZX-РЕВЮ" (N12, 1991г., с.227), мы же здесь рассмотрим этот вопрос в минимальном объеме хотя бы потому, что работая в БЕЙСИКе, Вы можете и не задумываться о потоках и каналах, а вот программируя в машинных кодах, без них не обойтись, коль скоро речь идет о графике, а значит о способах выдачи информации на экран.

Стандартными каналами "Спектрума" для вывода информации являются каналы:

- "K" - нижние две строки экрана (системное окно).
- "S" - главная часть экрана;
- "P" - стандартный ZX-принтер.

К этим каналам стандартно подключены потоки:
- поток #0 - к каналу "K";
- поток #1 - тоже подключен к каналу "K";
- поток #2 - подключен к каналу "S";
- поток #3 - к каналу "P"

Таким образом, оказываются идентичными следующие команды ввода/вывода:

PRINT #0 "Hello"; A$ - то же самое, что и INPUT "Hello": A$.

PRINT "Hello" - то же самое, что и PRINT #2 "Hello".

LPRINT "Hello" - то же самое, что и PRINT #3 "Hello".

Номер, стоящий после знака и в вышеприведенных примерах, является номером потока. Поскольку эти потоки подключены стандартно и переподключены быть не могут, мы программируем на БЕЙСИКе и используем операторы INPUT, PRINT, LPRINT без указания номера потока.

Это то, что касалось стандартных каналов и потоков, но они могут быть и нестандартными. Так, если Вы работаете в локальной сети, то сеть становится еще одним каналом, к которому вы подключите поток.

Вы знаете, что "Спектрум" может в любой момент времени выполнять только одно дело. Например, либо он печатает на экране, либо на принтере, одновременно выдавать информацию и туда и туда он не может, поэтому в любой момент времени задействован только один канал ввода/вывода и, соответственно, только один поток, связанный с ним. Этот канал и этот поток называются текущими. В большинстве случаев, когда Вы работаете с компьютером, текущим является канал "K", несколько реже канал "S".

Программируя в машинном коде, переключаться с канала "K" на "S" и наоборот - очень просто. О том, какой канал является текущим в данный момент, несет информацию нулевой бит системной переменной TVFLAG (5C3CH - 23612). Когда он выключен (равен 0), используется канал "S", а когда включен - "K".

Две небольшие процедуры продемонстрируют разницу в их использовании (листинг 1).

Не менее просто переключаться с каналов "S" или "K" на канал "P". Здесь тоже достаточно изменить один бит. Это первый бит системной переменной FLAGS (5C3BH - 23611). Он должен быть выключен для каналов "S" и "K", но включен для канала "P" (листинг 2).

Вы можете также изменить текущий канал, переключившись на другой поток. Это можно сделать вызовом процедуры ПЗУ, называющейся CHAN_OPEN и находящейся по адресу 1601H (5633). Перед тем, как ее вызывать, следует в аккумуляторе установить номер желаемого потока (листинг 3).

Нам необходимо знать эти азы потому, что если мы используем для печати из машинного кода команду процессора RST 10H, то должны иметь в виду, что она выдает информацию ТОЛЬКО В ТЕКУЩИЙ КАНАЛ. Прежде, чем Вы дадите компьютеру команду, что бы Вы хотели, чтобы он напечатал, надо сначала определиться, куда он будет это печатать и как.

Команду RST 10H Вы можете использовать для печати любых символов, будь то символ стандарта ASCII или графический символ. Это может быть управляющий символ и даже токен ключевого слова стандартного БЕЙСИКа. Поместите код того, что хотите напечатать, в аккумулятор и дайте команду RST 10H. При этом обычные символы займут одно знакоместо, управляющие коды сделают то, что им положено, а токены ключевых слов будут развернуты и займут столько знакомест, сколько букв в этом ключевом слове. Вам надо также знать, что команда RST 10H никогда не портит содержимое регистров процессора, кроме BC', DE' (альтернативные) и аккумулятора, который портит не всегда.

Печать чисел.

1. Целые числа от 0 до 9.

Выполнить печать целого числа от 0 до 9 можно двумя способами.

Во-первых, вы можете напечатать его обычным способом, как и любой другой символ. Для этого поместите в аккумулятор процессора его код (код 0 - 48 (30H).... код 9 - 57 (39H)) и дайте команду RST 10H.

Во-вторых, можно это число напечатать, загрузив в аккумулятор не его код, а само число, но в этом случае печать следует выполнять не командой RST 10H, а вызовом специально предназначенной процедуры ПЗУ - CALL 15EFH (десятеричный адрес - 5615), она называется OUT_ CODE. Это, конечно удобнее, но при своей работе эта процедура портит регистр E (имейте это в виду).

Возможен и промежуточный вариант, когда Вы засылаете в аккумулятор само число, а не его код, затем прибавляете к нему 30H (получаете его код) и затем даете RST 10H.

```
LD A,n
ADD A, 30H
RST 10
```

По расходу памяти это то же самое, что и CALL OUT_CODE, но не портит регистр E.

2. Целые числа от 0 до 9999.

Для печати целых чисел, меньших чем 10000, введите это число в регистровую пару BC и вызовите процедуру ПЗУ OUT_NUM_1. Она находится по адресу 1A1BH (6683).

Если Ваше число содержится в виде двух байтов в известном Вам адресе памяти, то Вы можете воспользоваться процедурой ПЗУ OUT_NUM_2 (1A28H = 6696). В этом случае перед вызовом процедуры надо в регистровую пару HL заслать адрес, в котором находится Ваше число.

И в том и в другом случае, если Вы попробуете через эти процедуры распечатать число, которое больше 9999, результат будет неверным.

Листинг 1.

Демо_S

```
213C5C          LD HL,TVFLAG
3600            LD (HL),00        ; Выключили бит 0 системной переменной TVFLAG.
3E2A      LOOP  LD A,42           ; Загрузили в аккумулятор число 42 (код символа "*")
D7              RST 10H           ; Выдали на печать по текущему каналу то,
                                  ; что находится в аккумуляторе.
18FB            JR LOOP           ; возврат для повтора.
```

Демо_K

```
213C5C          LD HL,TVFLAG
3601            LD (TVFLAG),01    ; Включили бит 0 системной переменной TVFLAG.
3E2A      LOOP  LD A,42
D7              RST 10H           : Печать символа "*".
18FB            JR LOOP           ; возврат для повтора.
```

Листинг 2.

Демо_P

```
213B5C          LD HL,FLAGS
CBCE            SET 1,(FLAGS)     ; Включили бит 1 системной переменной FLAGS.
0600            LD B, 00          ; Обнуление счетчика
                                  ; (подготовка к печати
                                  ; 256-ти символов).
3E2A      LOOP  LD A,42
D7              RST 10H           ; Печать символа "*".
10FB            DJNZ LOOP         ; Возврат для повтора,
                                  ; пока счетчик не достигнет нуля.
C9              RET               ; Возврат в вызывающую программу
```

3. Целые числа от 0 до 65535.

В этом случае Ваше число тоже должно быть помещено в регистровую пару BC, но в

отличие от предыдущего случая необходимо делать не один вызов процедур ПЗУ, а два.

Сначала оно должно быть конвертировано в интегральную (пятибайтную) форму и помещено на стек калькулятора, это делается вызовом процедуры STACK_BC (2D2BH = 11563). И только после этого оно может быть напечатано, как действительное число с плавающей точкой. Это выполняется вызовом процедуры PRINT_FP, расположенной по адресу 2DE3H (11747).

4. Отрицательные целые числа.

Знак "минус" имеет код 2DH. Поместите его в аккумулятор, выполните RST 10H и абсолютную величину числа печатайте, как показано выше.

5. Произвольные действительные числа (числа с плавающей точкой).

Такое число занимает 5 байтов и хранить его ни в каком регистре, ни в регистровой паре невозможно. В этом случае Вами должны быть приняты меры для того, чтобы предварительно разместить его на вершине стека калькулятора. Когда это сделано, вызов процедуры PRINT_FP (2DE3H=11747) напечатает его на экране.

Здесь надо сделать пару предупреждений для тех, кто работает с калькулятором "Спектрума", зашитым в ПЗУ.

Во-первых, после работы процедура PRINT_FP, число со стека калькулятора снимается и, если Вам оно еще может потребоваться, то позаботьтесь предварительно продублировать вершину стека (соответствующая команда в сводке команд калькулятора имеется - см. "Первые шаги в машинном коде". М.: "ИНФОРКОМ", 1990г., т. 1).

Во-вторых, в процедурах ПЗУ, обслуживающих калькулятор, есть ошибка. Она заключается в том, что это число при вызове PRINT_FP снимается со стека не всегда. Если Ваше действительное число находится в диапазоне от -1 до +1 (ноль исключается), то вместо вашего числа на вершине стека остается 0. Обращайте на это внимание.

Печать символьных строк.

У Вас есть по крайней мере три способа печатать текстовые сообщения, если вы работаете в машинном коде. Но во всех случаях этот текст должен храниться в памяти компьютера и начинаться с известного Вам адреса.

Самый простой метод состоит в следующем. Регистровая пара DE должна содержать адрес, с которого начинается ваша символьная строка, а в регистровой паре BC необходимо предварительно установить длину этой строки. Печать выполняется вызовом процедуры PR_STRING, которая находится по адресу 203CH (8252).

Во-вторых, символьная строка может быть Вами получена в результате работы встроенного калькулятора. В этом случае она находится на вершине стека и может быть напечатана прямо оттуда вызовом процедуры ПЗУ PR_STR_1 (адрес 2036H-8246).

Третья возможность - самая мощная и именно она применяется в большинстве случаев в игровых, прикладных и системных программах.

У Вас может быть целый набор из N различных сообщений, хранящихся в виде таблицы. И Вы, допустим, хотите напечатать k-ое сообщение. Тогда можете действовать следующим образом:

- установите в аккумуляторе число k-1:
- установите в регистровой паре DE адрес, указывающий на байт, находящийся перед первым байтом самого первого сообщения из Вашей таблицы сообщений. Имейте в виду, что этот байт должен быть больше или равен 80H, т.е. старший (седьмой) бит в этом числе должен быть включен (он явится маркером начала текстового сообщения);
- вызовите процедуру PO_MSG (0C0AH-3082) и Ваше сообщение будет напечатано на экране.

Впрочем, у этого метода есть ряд ограничений и требований. Их необходимо также иметь в виду при программировании:

- недопустимо использование графических символов или токенов ключевых слов БЕЙСИКа, т.е. коды печатаемых символов должны быть менее 128 (80H). Впрочем, можно ведь и поменять символьный набор, заменив на время символы ASCII на нужные вам графические шаблоны;

- во-вторых, Вам надо при заполнении таблицы сообщений сделать так, чтобы последний символ каждого сообщения имел включенный 7-ой бит, тем самым он будет служить маркером конца 1-го сообщения и компьютер всегда по номеру, заданному в аккумуляторе, найдет нужное Вам сообщение;

- в таблице не может быть пустых строк.

На что надо обратить внимание!

Работая с графикой из машинного кода, Вам надо иметь в виду некоторые особенности, связанные с работой встроенного калькулятора. Дело в том, что он не вполне свободен от разнообразных ошибок и неточностей, а они могут доставить головную боль начинающему программисту. Совсем другое дело, когда он предупрежден.

Итак, во-первых, если вам необходимо будет выдавать на экран символы блочной графики (коды с 80H по 8FH) - эти символы расположены на цифровом ряду клавиатуры, - то коррумпируются (портятся) системные переменные, расположенные в адресах с 5C92H (23698) по 5C99H (23705).

Те, кто читал "Первые шаги в машинном коде", знают, что здесь хранятся две первые ячейки памяти встроенного калькулятора (из шести стандартных). Это ячейки M0 и M1. Почти во всех случаях вам эта порча безразлична, но если вы активно работаете с калькулятором и именно они Вам и нужны, то примите меры по сохранению их значений в другом месте, например на стеке или в других ячейках.

Во-вторых, при печати чисел мы довольно активно пользовались процедурой ПЗУ PRINT_FP, которая распечатывает содержимое вершины стека калькулятора, а она во время своей работы "портит" все шесть ячеек памяти калькулятора, т.е. все содержимое системной переменной MEMBOT с адреса 5C92H (23698) по адрес 5CAFH (23727). Опять же для Вас это почти всегда безразлично, за исключением тех редких случаев, когда вы оставили там на хранение данные без присмотра.

А вот третий случай довольно часто встречается у тех, кто много работает со встроенным калькулятором. Дело в том, что в таблице системных переменных есть переменная под названием MEM, она расположена в адресе 5C68H (23656) и занимает 2 байта. В ней хранится адрес, по которому расположена память калькулятора, т.е. адрес системной переменной MEMBOT. Если Вам достаточно тех шести ячеек памяти, которые есть стандартно, то нет проблем. А если же Вам их мало, то Вы создаете их побольше, для чего меняете адрес MEMBOT и, естественно, указание на нее, содержащееся в MEM. Это нормальная, часто встречающаяся операция. Так вот, после такой замены процедура PRINT_FP правильно работать не сможет.

Управляющие символы.

Набор символов "Спектрума" включает в себя стандартные символы ASCII - их коды с 20H (32) по 7FH (127). Символы блочной графики, графики пользователя и токены ключевых слов БЕЙСИКа - коды с 80H (128) по FFH (255). Это оставляет вне поля нашего зрения символы с 0 по 1FH (31). Что же расположено там?

А здесь расположены так называемые управляющие символы, их еще называют управляющими кодами (символами их называть и не хочется, ведь их нельзя напечатать и увидеть). Хоть сами они на экране и не воспроизводятся, зато с их помощью можно управлять процессом печати тех символов, которые могут быть напечатаны.

Мы начнем с управляющего кода 16H. Он называется AT_CONTROL и определяет координаты позиции печати так же, как и оператор AT в БЕЙСИКе.

Задействуются управляющие коды, как и любые другие символы, путем выставления этого кода в аккумуляторе процессора и подачей команды RST 10H. Это означает, что они могут быть как бы "напечатаны" вместе с прочими символами и могут быть включены в длинные символьные строки.

БЕЙСИКовский оператор AT требует после себя указания двух параметров - координат позиции печати по вертикали и горизонтали. Управляющий код AT требует то же самое. Оба параметра вводятся тем же способом - вводом параметра в аккумулятор и выдачей команды RST 10H.

Приведенная распечатка показывает машиннокодовый аналог оператора БЕЙСИКа PRINT AT 5,1; .

DEMO AT_5_4

```
3E16  LD A, 16H
D7    RST 10H    ; PRINT AT ...
3E05  LD A,05
D7    RST 10H    ; 5......
3E04  LD A,04
D7    RST 10H    ; 4.....
```

Не имеет никакого значения, сколько и каких машиннокодовых команд будет выдано между тремя вышеприведенными командами RST 10H. "Спектрум", если выдал управляющий код 10H, далее всегда будет помнить, что два ближайших числа, проходящих через RST 10H, являются параметрами управляющего кода AT_CONTROL.

Более часто этот управляющий код находит применение, когда речь идет не о печати в фиксированной позиции, а в позиции, заданной программными переменными. Ниже показан машиннокодовый аналог команды PRINT AT D, E (здесь D и E - содержимое соответствующих регистров процессора.)

DEMO AT_D_E

```
3E16  LD A,16H
D7    RST 10H    ; PRINT AT ...
7A    LD A,D
DT    RST 10H    ; D, ....
7B    LD A,E
D7    RST 10H    ; E. ....
```

Следующий управляющий код, который мы рассмотрим - это код 17H. Он называется TAB_CONTROL - код управления табуляцией. До некоторой степени он тоже аналогичен оператору БЕЙСИКа TAB и указывает на номер позиции печати в текущей строке.

За ним также следуют 2 параметра (это не ошибка - именно 2 параметра, хотя БЕЙСИКовский опыт учит, что для позиции печати достаточно и одного). Дело в том, что первый параметр - это младший байт координаты позиции печати, а второй параметр - ее старший байт. Поскольку экран "Спектрума" имеет всего лишь 32 символа по ширине, сразу и не понятно, зачем еще нужен какой-то старший байт? Но дело в том, что печать ведь может идти не только на экран. А если на принтер? Да, ZX-принтер тоже имеет только 32 символа в строке, но ведь возможна печать и на широкий матричный принтер, через интерфейс. Внимательный читатель может спросить, где мы видели принтер, печатающий в строку более 255 символов, но это уже вопрос риторический. Неважно, есть он или нет, но возможность работы с ним в компьютере предусмотрена. Более того, понятие "печать", как способ выдачи информации, может предусматривать не только стандартные каналы типа экрана или принтера. Возможны ведь и пользовательские каналы типа файлов на диске или в оперативной памяти, в которых запись может иметь и десятки тысяч символов в строке.

Но мы имеем дело все же с экраном, поскольку речь идет о графике, и для печати на экране то, что находится в старшей байте, не имеет никакого значения.

Дело в том, что когда Вы задаете параметр TAB, компьютер берет из него остаток от деления на 32, а старший байт кратен 256 и, естественно, кратен 32, поэтому его содержимое влияние при печати на экран и не оказывает.

Нижеприведенная процедура показывает управляющий код TAB_CONTROL в действии. Первый параметр предполагается переменным и берется из регистра E. Выдавая второй параметр, мы обнулили его, использовав для этого обнуление аккумулятора через XOR A, но раз этот байт роли не играет, можно было XOR A и не давать, а отправить в этот параметр то, что было в аккумуляторе к этому моменту, неважно, что это было.

```
3E17  LD A,17
D7    RST 10H    ; PRINT TAB...
7B    LD A,E
D7    RST 10H    ; E....
AF    XOR A
D7    RST 10H    ; 0 ...
```

Код CHR 6 называется COMMA_CONTROL и выполняет то же, что и запятая оператора PRINT. Код подается точно так же, как и прочие, введением числа 6 в аккумулятор и выдачей его через RST 10H. В результате его действия в качестве позиции печати устанавливается 0 или 15, в зависимости от того, в какой половине экрана печаталось последнее знакоместо.

Код 0DH (13) - аналогичен ENTER. Выдается так же и обеспечивает прекращение печати в текущей строке и переход в начало следующей.

Код 08 - называется BACKSPACE_ CONTROL, он выполняет смещение позиции печати на одно знакоместо влево.

Команда процессора RST 10H -довольно гибкая и имеет самое разнообразное действие. С ее помощью можно не только управлять позицией печати символов на экране, но и управлять цветами того, что Вы воспроизводите, т.е. выдавать коды цветовых атрибутов. Выполняется это точно также путем предварительной установки в аккумуляторе управлявшего кода и передачей его, а затем передачей параметра. Мы не будем подробно останавливаться на цветовых атрибутах и привели их в Таблице 1.

Таблица 1.

Управляющий код	Параметры	Комментарии
06 COMMA_CONWR 08 BACKSPACE 0D ENTER	- - -	
10 INK_CONTROL 11 PAPER_CONTR	00-черный 01 -синий 02 -красный 03-пурпурн. 04- зеленый 05- голубой 06 -желтый 07-белый 08 -транспарантный 09 -контрастный	Параметр 08 (транспарантный[1]) означает, что цвет печати не устанавливается, а берется тот, который уже есть в данной позиции печати. Параметр 09 (контрастный) устанавливает цвет INK или PAPER контрастным к уже существующему в данной позиции противоположному цвету.
12 FLASH_CONTROL 13 BRIGHT_CONTR	00 - выключен 01 -включен 08 - транспарантный	См. выше.
14 INVERSE_CONTR 15 OVER_CONTROL	00- выключен 01-включен	
16 AT_CONTROL 17 TAB_CONTROL	Строка, столбец. Младший байт, старший байт.	При выдаче на экран старший байт может быть любым. Можете, например, использовать эту ячейку памяти для хранения какой-либо однобайтной переменной.

Пример применения кодов управления цветовыми атрибутами показывает, как выполняется печать красной звездочки на желтом фоне. Обратите внимание на то, что установленные таким способом цветовые атрибуты остаются действующими до того, как

[1] Transparent - прозрачный (Прим. OCR)

будут изменены или до окончания действия оператора БЕЙСИКа, из которого вызывалась данная процедура в машинных кодах.

```
3E10  LD A,10H
D7    RST 10H     ; PRINT INK...
3E02  LD A, 02
D7    RST 10H     ; 2; ....
5E11  LD A,11H
D7    RST 10H     ; PAPER...
3E06  LD A, 06
D7    RST 10H     ; 6;....
3E2A  LD A, 2A
D7    RST 10H     ; "*";
```

Другие приемы управления позицией и цветом печати.

В "Спектруме" есть возможность задействовать некоторые процедуры ПЗУ и управлять системными переменными для того, чтобы достигать тек же эффектов, которые дает применение управляющих кодов. Рассмотрим эти приемы.

PRINT AT B,C (где B и C -содержимое одноименных регистров процессора) может быть выполнено вызовом процедуры AT_B_C, находящейся по адресу 0A9BH (2715).

PRINT TAB A (где A - содержимое аккумулятора) выполнимо вызовом процедуры TAB_A (0AC3H=2755).

Управляющий код 06 (COMMA_CONTROL) эмулируется вызовом PO_COMMA (0A5FH=2655).

Эти три приема действуют не только на канал "S", но и на канал "K", т.е. с их помощью можно печатать информацию и в INPUT-строке.

У Вас есть также достаточно простой способ управлять цветовыми атрибутами INK, PAPER, BRIGHT, FLASH. Все, что для этого требуется - внести изменения в системную переменную ATTR_T (5C8FH = 23695), отвечающую за статус временных атрибутов экрана, эта системная переменная имеет следующую раскладку:

Биты 0...2 - цвет INK (от 0 до 7)

Биты 3...5 - PAPER (от 0 до 7)

Бит 6 - статус BRIGHT (0 или 1)

Бит 7 - статус FLASH (0 или 1)

Все, что требуется для установки нужных комбинации цвета - это заслать в эту системную переменную число, определяемое по формуле:

128*F + 64*B + 8*P + 1, где:

- F - статус FLASH:
- B - статус BRIGHT;
- P - номер цвета PAPER;
- I - номер цвета INK.

Если же Вы хотите, чтобы какие-то цветовые атрибута были транспарантными, системной переменной ATTR_T Вам уже недостаточно и надо воспользоваться системной переменной MASK_T (5C90H = 23696). Ее раскладка точно та же, что и у ATTR_T. Биты, соответствующие атрибуту, который Вы хотите сделать транспарантным, надо включить.

Итак, рассмотрев, каким образом из машинного кода выполняется печать графики низкого разрешения, мы переходим к графике высокого разрешения, но для этого надо сначала хорошо разобраться со структурой экрана и, главное, понять, как раскладка экрана связана с организацией экранной памяти. Отсюда мы сделаем шаг к наиболее рациональным приемам по изменению экранных форм из машинного кода.

Организация экранной памяти.

Область оперативной памяти "Спектрума", в которой находится графическое изображение, видимое на экране в качестве включенных и выключенных точек (пикселов), называется экранной областью памяти.

Эта область занимает адреса с 4000H по 5AFFH (16384 - 23295). Причем, она состоит

из двух областей. В первой с 4000H по 57FFH (с 16364 по 22537) расположено растровое изображение, состоящее из черных и белых неокрашенных точек (дисплейный файл). Во второй, с 5800H по 5AFFH (с 22528 по 23295) - цветовая информация (файл атрибутов).

Размер всей экранной области - 6912 байтов. Из них 6144 байта - дисплейный файл (черно-белый) и 768 байтов - файл атрибутов (цвет).

Как получены эти числа 6144 и 728?

Вы знаете, что полный экран "Спектрума" может иметь 24 ряда по 32 символа в ряду, т. е. всего 24*32=728 знакомест. Каждое знакоместо образовано из 64-х пикселов (8 линий по 8 точек). На каждую линию достаточно одного байта (8 битов соответствуют 8 точкам) и, следовательно, для растровой графики одного знакоместа необходимы 8 байтов. Так, на 728 знакомест необходимо 728*8=6144 байта для хранения черно-белой информации.

Информация о цвете хранится более экономно. Каждому знакоместу отдан один байт, определяющий окраску всех 64 его пикселов. Три младших бита определяют цвет INK этого знакоместа, еще три бита определяют цвет PAPER и по одному биту отдано признакам BRIGHT и FLASH. Отсюда вытекает одно из самых неприятных ограничений графики "Спектрума" - в пределах одного знакоместа невозможно иметь одновременно более 2-х цветов.

А теперь давайте немного поэкспериментируем. Очистите экран - CLS. Окрасьте бордюр в голубой цвет - BORDER 5. Это нужно, чтобы точно видеть результат следующих манипуляций и дайте команды, задающие самый первый байт экранной области:

```
POKE 16384,255
POKE 16384,15
POKE 16384,85
```

После каждой команды вы увидите изменения в левом верхней углу экрана, а точнее - в первой линии первого знакоместа. На рис. 1 показана эта линия укрупненно и Вам должно быть понятно, что и почему там происходит. Вы видите, что раскладка пикселов в этой линии соответствует раскладке битов в байте 16384, отвечающем за эту линию.

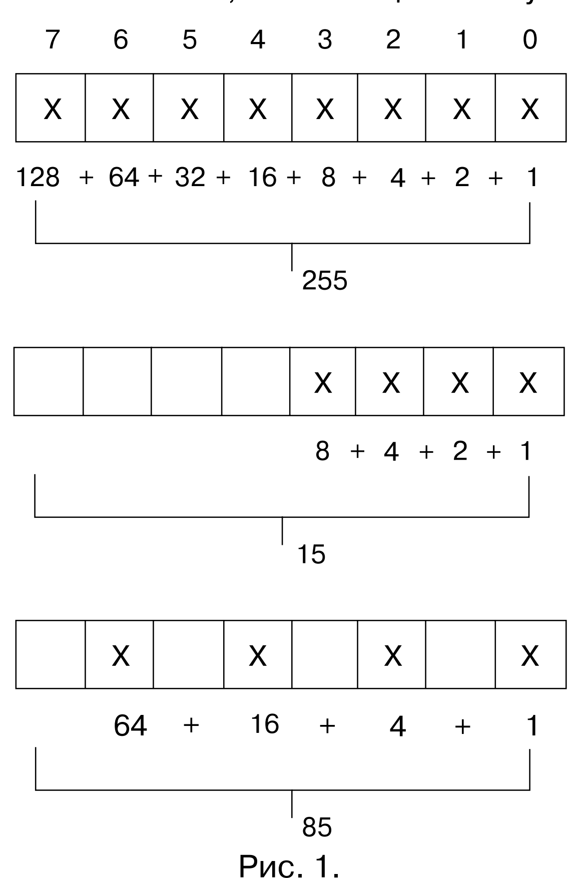


Рис. 1.

Теперь попробуем закрасить вторую линию в первом знакоместе. Логично предположить, что POKE в следующую ячейку памяти сделает это. Дайте POKE 16385,85. Получилось совсем не то - включилась первая линия второго знакоместа, далее третьего и так далее, до 32-го. Может быть, попробуем их пропустить и дадим POKE сразу в 33-ий адрес:

```
POKE (16384*32), 85
```

И опять ничего не получилось. Вместо второй линии в первом знакоместо включается первая линия в первом знакоместе, но во втором ряду. И так будет продолжаться, пока Вы не пройдете все 32 знакоместа в 8 рядах экрана, т.е. 32*8=256 адресов. Только дав POKE (16384 + 256), Вы получите то, что хотели. На первый взгляд, такое соответствие между точками экрана и ячейками экранной памяти выглядит достаточно нелепым и нелогичным. Но это не совсем так. Как окажется чуть ниже, это не только не усложняет жизнь, а при работе из машинного кода даже упрощает - надо только хорошо все понять.

Теперь, покопавшись в памяти, Вы наверняка вспомните, что уже много раз видели при загрузке заставок игровых программ, как экран прорисовывается постепенно, строчка за строчкой. В этот момент Вы и видели соответствие между структурой экрана и экранной памятью. Хотите повторить, пожалуйста:

```
10 CLS: BORDER 5
20 FOR i=16384 TO (16384+256*8)
30 POKE i, 255
40 NEXT i
```

Чтобы не утомлять Вас длительным ожиданием, мы сделали это только для 8 первых рядов экрана.

Научившись изображать на экране точки и тире без помощи операторов PRINT и PLOT, мы уже сделали большое дело и развязали себе руки, т.к. мы теперь умеем манипулируя с ячейками памяти делать графику, а машинный код именно и хорош, когда дело доходит до манипуляций с большими объемами памяти.

Но как насчет того, чтобы получить нормальный графический образ? Нет проблем, этот образ сначала надо где-то в памяти создать. Давайте воспользуемся готовым. Вы, конечно знаете, что в ПЗУ компьютера где-то имеется набор символов, в котором хранятся графические образы (шаблоны) всех букв и прочих знаков, вот мы возьмем оттуда образ буквы "А" и нарисуем ее на экране без PLOT или PRINT.

В адресах 23606, 23607 (5С36Н) хранится 2-х байтная системная переменная CHARS, которая указывает на 256 байтов ниже, чем адрес, с которого начинается набор символов. Во-первых, давайте разберемся, почему она указывает ниже на 256 байтов, а не туда, куда надо. Все очень просто. Мы уже говорили о том, что первые 32 символа вовсе и не символы, а управляющие коды и им графические образы не нужны, все равно они не печатаются. А т.к. символы занимают по 8 байтов каждый, то на пропуск этих 32 управляющих кодов и ушло это снижение на 256 байтов. Зато теперь мы можем искать образ буквы А по ее номеру (по ее коду ASCII, который, кстати, равен 65(41Н)).

```
10 LET base=PEEK 23606 + 256*PEEK 23607
20 LET addr = base+8+65
25 LET screen= 16384
30 FOR i=0 TO 7
40 LET pic=PEEK (addr+i)
50 POKE (screen+256*i), pic
60 NEXT i
```

Мы с Вами только что нарисовали букву "А", причем именно нарисовали, а не напечатали, да к тому же работали при этом только с переброской данных из одних участков памяти в другие. Вы обратили внимание на то, что засылали мы графику в дисплейный файл в строке 50 с шагом через 256 адресов? Давайте теперь рассмотрим, как то же самое будет выглядеть в машинном коде и почему так сложно, на первый взгляд, организован дисплейный файл.

Адреса экранной памяти обычно при работе с экраном хранятся в регистровой паре HL. При этом в Н хранится старший байт адреса (High - старший), а в L - младший (Low), для того, чтобы увеличить адрес на 256 и перейти к нижележащей линии в том же знакоместе, оказывается достаточно увеличить на единицу старший байт. С этой задачей изящно справляется простая команда АССЕМБЛЕРа INC H. Увеличение же на единицу младшего байта адреса (INC L) переместит Вас на соседнее знакоместо вправо в той же линии. Видите, как все просто, и преобразование вышеприведенной БЕЙСИК-программы в

машинный код выглядит (листинг 4):

ЛИСТИНГ 4.

```
        LD DE,(23606)          ; Загрузили в DE содержимое системной переменной CHARS.
        LD HL,0041H            ; Загрузили в HL код буквы А.
        ADD HL,HL              ; Умножили его на 2.
        ADD HL,HL              ; Умножили его на 4.
        ADD HL,HL              ; Умножили его на 8.
        ADD HL,DE              ; Теперь HL указывает на начало шаблона буквы А.
        EX DE,HL               ; Освободили HL для работы с экраном.
        LD HL,4000H            ; Загрузили в HL адрес начала экранного файла.
        LD B,08                ; Организуем счетчик на 8 шагов
 LOOP   LD A,(DE)              ; Переброска содержимого из
        LD (HL),A              ; DE в экранный файл через аккумулятор.
        INC H                  ; переход к новой линии экрана.
        INC DE                 ; Переход к новой линии шаблона.
        DJNZ LOOP              ; Окончание цикла из 8-ми шагов.
        RET                    ; возврат.
```

Итак, все вроде бы просто и понятно, но Вы, конечно, обратили внимание на то, что все, что мы до сих пор делали, относится только к восьми первым символьным рядам экрана, а что же дальше? Ведь их всего 24.

Здесь тоже все просто, если представить себе, что экран состоит из трех несвязанных между собой областей, каждая из которых имеет по 8 рядов. Назовем эти трети экрана сегментами. Итак, экран состоит из трех сегментов, каждый из которых состоит из восьми рядов, каждый из которых состоит из тридцати двух знакомест, каждое из которых состоит из восьми линий, каждая из которых состоит из восьми пикселов.

Первый сегмент -16384 - 18431 (4000H - 47FFH)

Второй сегмент -18432 - 20479 (4800H - 4FFFH)

Третий сегмент - 20480 - 22527 (5000H - 57FFH)

Каждый сегмент занимает по 8*32*8=2048 байтов.

Если теперь развернуть регистровую пару HL в виде шестнадцати битов, то получится любопытная побитовая карта для дисплейного файла (см. рис. 2).

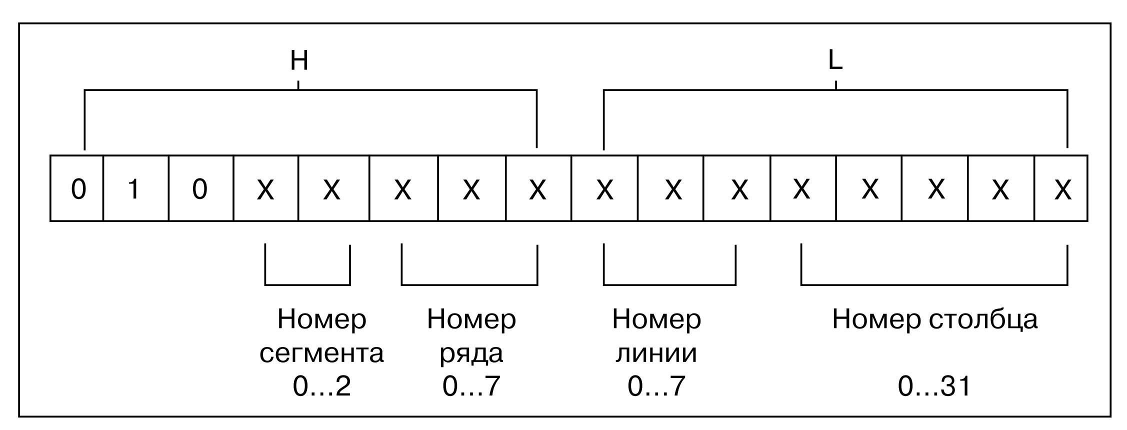


Рис.2

Однажды, в 1985 году сэра К. Синклера спросили, чем он объясняет столь невероятный успех своих компьютеров по сравнению с главными конкурентами "Коммодором", "Атари", "Амстрадом" и "Би-Би-Си Микро", если известно, что графика у них лучше, музыка богаче, надежность выше и дополнительных внешних портов больше? На это он ответил: "У меня гораздо проще доступ к памяти". И этим сказано все. В частности, организация экранной памяти в "Спектруме" была не последним фактором, обеспечившим ему поддержку со стороны сотен фирм, выпускающих программное обеспечение.

Давайте рассмотрим эти преимущества.

1. В отличие от многих других разработчиков К. Синклер нашел удачное решение, объединив в едином экране и графику высокого разрешения и символьную графику.

Нет необходимости переключаться на другой экран всякий раз, как надо построить диаграмму или написать текст. Это большой подарок программистам.

2. Решение о разделении черно-белой информации высокого разрешения и цветовой информации низкого разрешения в разные файлы позволило ему в 4-5 раз уменьшить объем экранной памяти (по сравнению с некоторыми конкурентами) и выделить много места для пользователя. Практически здесь имитируется цветная графика высокого разрешения, хотя она таковой строго говоря и не является - это был гениальный ход, высоко оцененный специалистами.

3. Эти решения позволили в значительной степени сократить размеры программ ПЗУ. Действительно, по отношению возможностей ПЗУ к его размерам вряд ли найдется машина, равная "Спектруму", хотя и у этого ПЗУ, как показал дальнейший опыт, еще были огромные резервы.

Теперь посмотрим, как на практике ухватились программисты например за такой простой элемент, как сегментирование экрана на три зоны при том, что экран является одновременно и графическим и символьным.

Во-первых, множество первоклассных программ используют для своей графики только один сегмент, оставив прочее на диалог с пользователем. Ведь стоит только вдуматься, что в таких программах, как TIR-NA-NOG, DUN DARACH, MARSPORT фирмы "GARGOYLE GAMES" огромные события происходят на участке памяти размером всего лишь 2K.

Во-вторых, стал традиционным прием, при котором графика занимает вроде бы весь экран и непрофессионал даже и не замечает, что два сегмента из трех заняты красочной статической графикой, в то время как все действие разворачивается лишь на одном сегменте (может быть и менее красочном). Малый размер памяти этого сегмента позволяет достичь высокого быстродействия, плавности переметения объектов, прекрасной реакции на действия пользователя, а общий эстетический эффект непроизвольно распространяется на весь экран. Вроде бы имитация, но какая!

В третьих, очень часто экранную память неиспользуемых сегментов начинают использовать для разных вспомогательных действий, для размещения временных таблиц, буферов и т.п., втискивая в небольшие размеры оперативной памяти огромное количество информации. Вы сами, возможно видели во время работы копировщика COPY-86M, как временно ненужная информация выбрасывается на экран в виде точек и тире в двух нижних сегментах экрана. Это же происходит и при работе кнопки MAGIC BUTTON в операционной системе TR DOS. Тот же прием применяют и в игровых программах, но там Вы этого не увидите, т. к. программист заблаговременно установил в цветовых атрибутах, относящихся к этим сегментам экрана, черный цвет INK и черный цвет PAPER. За черным цветом прячут иногда даже и машинный стек процессора (например в программе NETHEREARTH), что не только экономит оперативную память, но является еще и эффективным приемом защиты программы от внешнего вторжения, и многое-многое другое.

Файл атрибутов.

Теперь, рассмотрев работу с дисплейным файлом, перейдем к файлу атрибутов и посмотрим работу следующей программы:

```
10 PAPER 6; INK 0: BORDER 6: CLS
20 FOR i=1 TO 22
30 PRINT "ZX-SPECTRUM"
40 NEXT i
50 FOR i=22528 TO 23295
60 POKE i,15
70 NEXT i
```

Посмотрите, что произойдет, когда Вы запустите эту программу. Во-первых, на экране будет напечатан текст (черным цветом по желтому фону). Во-вторых, знакоместо за знакоместом, начнется изменение его цвета (белые буквы по синему фону). Чтобы понять, почему так происходит, нам надо изучить область памяти от адреса 22528 до адреса 23295 (5800H -5AFFH), которая и называется файлом атрибутов. Содержимое ячеек памяти в этой области определяет то, каким цветом будет окрашено то или иное знакоместо.

Структура этого файла очень проста. Каждому знакоместу экрана соответствует один байт памяти, а значение этого байта и определяет цвета этого знакоместа. Это означает, что когда вы засылаете (POKE) какое-либо число в ячейку памяти этой области, Вы изменяете цвет одного из знакомест экрана. Каждый байт из файла атрибутов хранит информацию о параметрах INK, PAPER, BRIGHT и FLASH.

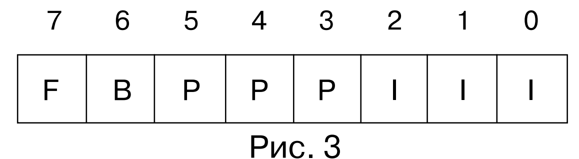
Биты 0...2 - цвет INK (от 0 до 7)

Биты 3... 5 - PAPER (от 0 до 7)

Бит 6 - статус BRIGHT (0 или 1)

Бит 7 - статус FLASH (0 или 1)

Схематически это выглядит так:



7	6	5	4	3	2	1	0
F	B	P	P	P	I	I	I

Рис. 3

Для программирующего в машинном коде, конечно, чрезвычайно важно знать точно, какому знакоместу экрана соответствует какой байт файла атрибутов. Зная, что в каждом ряду 32 знакоместа, этот адрес вычислить нетрудно:

22528 + 32 * Y + X

Здесь Y и X - координаты знакоместа. Y - номер ряда сверху вниз, X - номер столбца слева направо. Как видите, в отличие от дисплейного файла, файл атрибутов организован вполне благоразумно: слева направо и сверху вниз.

Изменение цвета бордюра.

При работе в машинном коде здесь есть небольшая сложность, заключающаяся в том, что для изменения цвета бордюра требуется два действия, т.к. к этой цели ведут два пути. Дело в том, что тот существует текущий цвет бордюра (тот, который Вы в данный момент видите на экране) и есть установленный цвет бордюра (тот, который записан в памяти компьютера и будет установлен, как только ПЗУ компьютера перехватит управление от Вашей программы в машинных кодах, например при нажатии какой либо клавиши).

Ваша задача состоит в изменении обоих цветов и текущего и установленного, т. к. изменение только текущего будет иметь кратковременный характер, а изменение только установленного вообще никак не отразится на экране в данный момент.

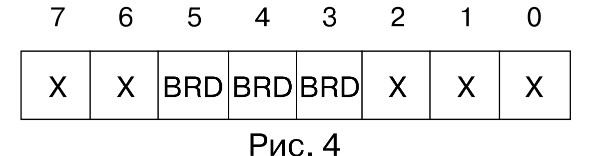
Текущий цвет бордюра изменяется выдачей байта по 254-ому внешнему порту (бордюрная часть экрана является для "Спектрума" внешним устройством, подключенным к порту FEH). Это можно сделать даже из БЕЙСИКа, воспользовавшись командой OUT:

```
OUT 254, цвет
```

В машинном коде эта команда выглядит удивительно похоже:

```
LD A, цвет
OUT (FEH),A
```

Установленный цвет бордюра, с которым работает ПЗУ, постоянно хранится в отведенной для него ячейке памяти в области системных переменных. Ее название BORDER. Ее адрес 23624 (5C48H). Три бита этой переменной отвечают за цвет бордюра:



7	6	5	4	3	2	1	0
X	X	BRD	BRD	BRD	X	X	X

Рис. 4

Прочие биты занимаются другими делами, в частности, они задают цвет нижней части экрана, используемой для работы INPUT и пр.

Оба переключения одновременно можно сделать вызовом процедуры ПЗУ BORDER_A, расположенной по адресу 2297H (8855). Предварительно код цвета надо установить в аккумуляторе процессора. У нее есть и побочное воздействие. Если при вызове этой процедуры что-то содержится в системных нижних двух строках компьютера, то она обойдется с ними достаточно "гуманно". Чтобы бордюр не скрыл информацию, цвет INK этих строк будет установлен контрастным к цвету бордюра.

Владельцам 128-х машин.

"Спектрумы" со 128 килобайтной памятью имеют не одну, а две экранных области. (См. "ZX-РЕВЮ", N1, с.4; N2, с.26; М.: ИНФОРКОМ, 1991), каждая из которых может быть использована для хранения изображения, если Вы работаете в режиме 128К.

Первая экранная область, как обычно занимает адреса 4000H-5AFFH, а вторая расположена в адресах C000H - DAFF на седьмой странице оперативной памяти. Первая область называется нулевым экраном, а вторая - первым экраном. Очевидно, что только одна область из двух может быть воспроизведена на экране телевизора в данный момент.

Раскладка экрана-1 точно такая же, как и экрана-0 и мы можем считать, что разница состоит во-первых, в 15-ом байте (рис. 5), а во-вторых в том, что он расположен не на нулевой странице ОЗУ, а на седьмой.

Тот экран, который в настоящее время задействован, называется активным. Обратите внимание на интересную особенность первого (дополнительного) экрана. После того, как он был задействован, уже нет необходимости седьмой странице ОЗУ оставаться "впечатанной". Т.е. экран-1 может оставаться активным независимо от того, какие страницы ОЗУ "впечатаны" в данный момент.

Есть еще одна особенность: процедуры ПЗУ могут работать только с нулевым (основным) экраном. Они не умеют печатать и рисовать на втором экране. Ни PLOT, ни PRINT, ни DRAW, ни даже машиннокодовая команда RST 10H не работают с первым экраном. С ним не работают ни автоматический листинг, ни система редактирования, ни INPUT.

"Спектрум-128", как это ни странно, не содержит в своем ПЗУ-128 совершенно ничего, чтобы переключаться с одного экрана на другой и пользователю приходится самому писать для этого процедуру, например такую, как показано ниже, точками входа в нее являются метки SCR_0 и SCR_1, соответствующие тому, какой экран Вы хотите вызвать (листинг 5).

Примечание: Обязательный порядок всех переключений в 128-х машинах - сначала перенастраиваем BANK_M и только потом выдаем команду по порту 7FFD. Системные прерывания компьютера вызывают запуск его процедур ПЗУ, которые, заканчивая работу, выдают по порту 7FFD содержимое BANK_M. Так что, если Вы будете делать наоборот, есть высокая степень вероятности, что у Вас ничего не получится, правда, из машинного кода прерывания можно и отключать (см. "Первые шаги в машинном коде").

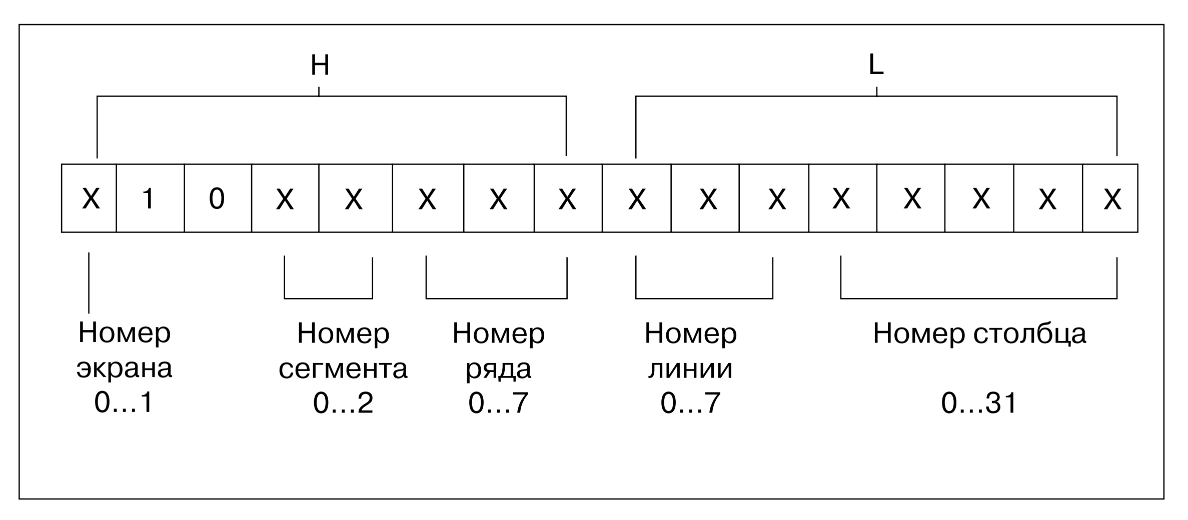


Рис. 5

ЛИСТИНГ 5.

```
1E00     SCR_0    LD E,00         ; Все биты выключены.
1802              JR SELECT       ; Переход на переключение экрана.
1E08     SCR_1    LD E,08         ; Включен бит-3.
3A5C5B            LD A,(BANK_M)   ; Системная переменная
                                  ; BANK_M (5B5CH) - служит
                                  ; в 128-килобайтных машинах
```

```
                              ; как указатель страниц
                              ; ОЗУ, ПЗУ, экрана и
                              ; режима (см. рис. 6).
                              ; За номер экрана отвечает
                              ; бит 3.
E6F7           AND F7         ; включили все биты в
                              ; аккумуляторе, кроме 3-го.
B3             OR E           ; третий бит включается
                              ; или выключается в зависимости
                              ; от того, через какую
                              ; точку мы вошли в
                              ; эту процедуру.
325C5B         LD (BANK_M),A  ; Переключили BANK_M на
                              ; экран-1. Но он еще не
                              ; активен. Это только подготовка.
01FDF7         LD BC,7FFD     ; Установили в BC номер
                              ; внешнего порта, служащего для
                              ; физического переключения страниц и режимов.
ED79           OUT (C),A      ; Переключение экрана.
C9             RET            ; Выход
```

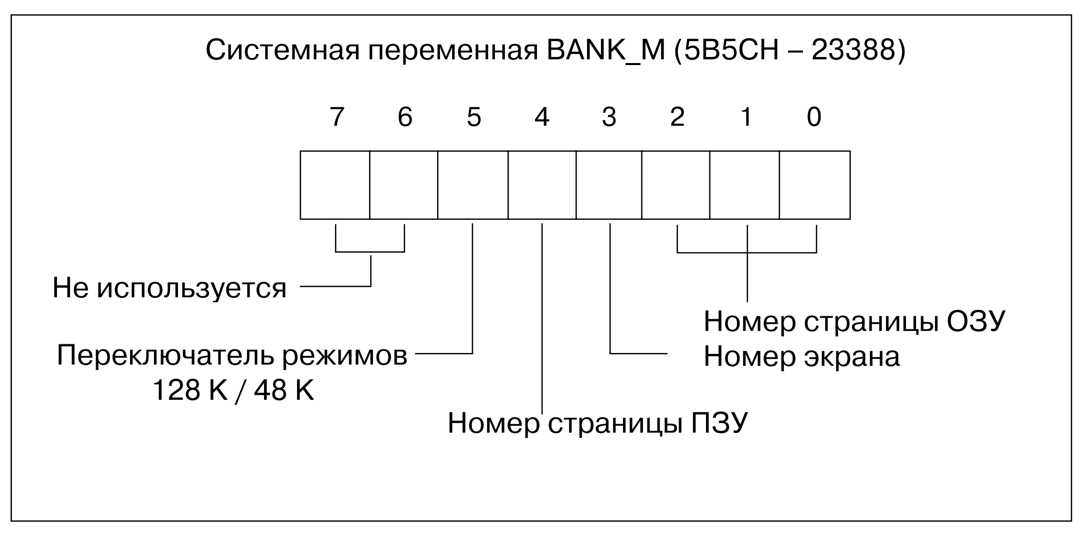


Рис.6

И еще одно предостережение для тех, кто работает с RAM-диском на 128-килобайтных машинах (о RAM-диске см. "ZX-РЕВЮ", N12,С.235-240; М.:ИНФОРКОМ, 1991.). Когда вы выгружаете что-либо в RAM-диск, используя для этого новую команду SAVE!, файлы сохраняются в памяти в виде стека - один над другим. Начало области - 1C000 и далее вверх до 1FFFF, потом с 3C000 по 3FFFF, с 4C000 по 4FFFF, с 6C000 по 6FFFF и, наконец, с 7C000 и вверх.

Одновременно с этим образуется второй стек. Он начинается в адресе 7EBFF и развивается вниз, В нем содержатся только имена и адреса файлов, выгруженных на RAM-диск. Короче говоря, этот стек содержит ту информацию, которую Вы получаете, воспользовавшись новой командой CAT! . Таким образом, эти два стека развиваются навстречу друг другу, встречаться они не должны. При появлении такой опасности компьютер выдаст сообщение об ошибке "4 Out of memory", стек каталога не может также опуститься ниже 7C000 - именно поэтому установлено ограничение на количество сохраняемых файлов - 562.

Итак, все сделано, чтобы эти стеки не встретились и не испортили друг друга, но ничего не сделано для того, чтобы они не затерли экран-1. Этот экран может быть испорчен как стеком файлов, так и каталожным стеком. И даже наоборот, работая с экраном-1, Вы можете испортить эти стеки. Будьте внимательны. Работая в машинном коде, использовать экран-1 можно только если Вы не очень активно используете виртуальный диск. Смело можете хранить до 216 файлов с суммарной длиной не более 64К.

Эмуляция команд БЕЙСИКа из машинного кода.

Здесь мы рассмотрим, как выполнить БЕЙСИК-овские команды, связанные с графикой, из машинного кода. Как правило, это несложная задача. Поскольку в ПЗУ имеются процедуры, способные это сделать, надо только знать, каким образом к ним следует обращаться.

CLS

Сначала займемся очисткой экрана. В машинном коде это можно сделать вызовом процедуры CLS, расположенной в ПЗУ по адресу 0D6BH (3435). В результате ее работы выключаются все пикселы на экране, а цветовые атрибуты устанавливаются такими, как установлено в системной переменной ATTR_P 5C8DH (23693). Ее побитовая раскладка та же, что и у системной переменной ATTR_T (см. выше). Данная процедура работает точно так же, как и соответствующая команда БЕЙСИКа.

Вы можете также очистить только нижнюю часть экрана (обычно это две нижние строки). Это делается вызовом процедуры CLS_LOWER, которая находится по адресу 0D6EH (3438). В отличие от основного экрана, эта область очищается с цветом бордюра, а не с цветом, установленным в ATTR_P.

Среди процедур ПЗУ есть еще одна, обеспечивающая более мощные возможности. Она называется CL_LINE (0E44H = 3652) и очищает заданное количество строк в нижней части экрана.

Перед ее вызовом в регистре B должно быть установлено количество строк, подлежащих очистке от 01 до 18H (от 1 до 24). Так, например, чтобы очистить 10 нижних строк, нужно предварительно в регистр B заслать число 0AH.

Эту процедуру можно применять двумя способами. Если нулевой бит системной переменной TVFLAG (5C3C = 23612) выключен, то при очистке используются цвета экрана, а если он включен, то используется цвет бордюра. Таким образом, здесь есть возможность очистить одновременно весь экран, включая и нижние две строки, в цвет ATTR_P. Процедура CLS сделать такого не может. В то же время, следует обратить внимание на то, что процедура CLS после очистки экрана инициализирует курсор и текущая позиция печати устанавливается в исходное положение - левый верхний угол. Процедура же CL_LINE этого не делает.

Скроллинг экрана.

Когда Вы пытаетесь напечатать что-либо за последней возможной позицией печати, то вместо результата получаете сообщение "scroll?" в нижней части экрана и компьютер ждет от Вас нажатие клавиши. Если это клавиша "N" или "BREAK", печать на экране прекращается с сообщением об остановке, в противном случае печать продолжается. Экран вроде бы заполнен, но печать возможна, происходит скроллинг вверх. Этот скроллинг происходит автоматически и длится до тех пор, пока вся информация, присутствовавшая на экране, когда выходило сообщение "scroll?", не скроется из виду за верхней кромкой экрана.

Такой вид скроллинга называется автоматическим и он существует на "Спектруме" как в БЕЙСИКе, так и в машинном коде.

Но возможен также и "ручной" скроллинг. Вызов процедуры CL_SC_ALL (0DFEH = 3582) "прокрутит" весь экран, но при этом возникнут несколько неожиданные эффекты:

- во-первых, позиция печати не изменится и останется там же, где была. С этим Вам придется разбираться вручную.

- во-вторых, если первая строка нижней части экрана (системного окна) не пуста или имеет цвет, отличный от цвета основного экрана, то результат может быть не совсем тем, какой Вы ожидаете.

Другой способ исполнения "ручного" скроллинга еще интереснее. Он базируется на использовании процедуры CL_SCROLL (0E00H 3584). С ее помощью можно "прокручивать" только часть экрана. Предварительно в регистр B надо занести количество строк, подлежащих скроллингу (обратите внимание на то, что изменено будет на одну строку

больше). Минимальное количество строк - 1. В этом случае нижняя строка экрана будет поднята на одно знакоместо вверх, а снизу появится чистая строка. Для того, чтобы переработать весь экран, следует установить число 17H. Таким образом, действие этой команды состоит в перемещении на одну позицию вверх "B" нижних строк и в очистке нижней строки.

PAUSE

БЕЙСИК-оператор PAUSE можно легко эмулировать в машинном коде. Единицей измерения времени в "Спектруме" являются пятидесятые доли секунды (потому, что частота переменного тока в нашей сети 50Гц. Если бы Вы жили в Англии, то у Вас в секунду проходило бы не 50, а 60 прерываний работы процессора, т. к. там частота сети равна 60 Гц[1]).

Для того, чтобы исполнить паузу на m пятидесятых долей секунды, Вам достаточно загрузить это число m в регистровую пару BC, а затем вызвать процедуру ПЗУ PAUSE_1, расположенную по адресу 1F3DH (7997). Работая, процедура обеспечит заданную паузу. Во время своей работы процедура регулярно опрашивает системную переменную FLAGS (5C3BH = 23611). Ее пятый бит, если он включен, свидетельствует о том, что произошло нажатие клавиши. Таким образом, пауза может быть прервана нажатием любой клавиши - все, как в стандартном БЕЙСИКе. Если Вы зашлете в регистровую пару BC ноль, то установленная пауза будет неограниченной, то есть до нажатия клавиши.

Вам, может быть, потребуется случай, когда пауза должна быть выдержана в течение заданного времени и не должна прерываться нажатием клавиши, поможет несложная процедура (листинг 6).

Зашлите в BC требуемую величину задержки и вызывайте эту процедуру.

Листинг 6.

```
78     PAUSE      LD A, B       ; Проверка BC на
B1                OR C          ; ноль.
C8                RET Z         ; Выход, если так.
0B                DEC BC        ; Уменьшение BC,
76                HALT          ; Задержка на 1/50
                                ; секунды (точнее - до
                                ; следующего прерывания,
                                ; которое пройдет
                                ; чуть быстрее).
18F9              JR PAUSE      ; Возврат для повтора.
```

Рисование точек.

Для изображения точек на экране из машинного кода есть два приема. Первый, и наиболее простой состоит в том, чтобы загрузить координату X в регистр C, а координату Y - в регистр B, а затем вызвать процедуру PLOT_SUB, расположенную по адресу 22E5H (8933). Если у Вас координаты позиции печати точки уже содержатся в системной переменной COORDS (5C7D = 23677), то входить в эту процедуру можно в точке 22E8H (8936). Обратите внимание на то, что когда процедуры ПЗУ вычисляют по координатам позиции печати адрес соответствующего байта в дисплейной памяти, они коррумпируют регистр HL. Если он Вам нужен, его надо сохранить на стеке и потом восстановить.

Другой метод может быть более приемлем для тех энтузиастов, которые работают со встроенным калькулятором. Обе координаты должны быть предварительно помещены на вершину стека калькулятора в порядке: сначала X, затем Y. Вызовом процедуры PLOT (22DCH = 8924) эти координаты снимаются со стека и точка печатается на экране.

---

[1] Это связано частотой кадровой развертки видеосигнала формата PAL, принятого в Великобритании (50 Гц). (Прим. OCR)

Рисование прямых линий.

Здесь также существуют два метода работы из машинного кода, но прежде, чем мы их рассмотрим, напомним, что в операторе БЕЙСИКа DRAW X, Y параметры X и Y не являются координатами экрана, а являются "смещением" точки конца отрезка относительно точки его начала. "Смещение" может быть не только положительным, но и отрицательным.

Метод 1.

"Смещение" X загружается в регистры C и E. При этом в регистре C устанавливается его абсолютная величина ABS (X), а в регистре E устанавливается 01, если "смещение" положительно или равно нулю, но FFH, если отрицательно. Аналогично "смещение" Y загружается в регистры B и D. В регистре B - ABS (Y), а в регистре D - 01 или FFH.

После этого можно рисовать линию, вызвав процедуру DRAW_LINE с точкой входа 24BAH (9402). Линия будет нарисована, но следует предусмотреть сохранение регистровой пары H'L' (альтернативная), т.к. она будет коррумпирована. Второй путь, как и для печати точек, состоит в использовании калькулятора. Смещения X и Y (в таком порядке) должны быть установлены на вершине стека калькулятора и тогда та же процедура DRAW_LINE, но с точкой входа 24B7 (9399) нарисует отрезок прямой на экране. H'L' здесь коррумпируется точно так же.

Рисование дуги.

Оператор Бейсика DRAW может вычерчивать между двумя точками не только прямые линии, но и соединять их с помощью дуги. Форма оператора - DRAW X,Y,A. Параметр A определяет угловую меру дуги, а X, Y - "смещения". Угловая мера "A" задается в радианах.

Чтобы выполнить аналогичную задачу из машинного кода, следует все три параметра поместить на стек встроенного калькулятора в порядке X, Y, A. Затем вызывается процедура DRAW_ARC (2394H = 9108). Дуга рисуется, как серия очень маленьких прямых, т.е. эта процедура в своей работе многократно вызывает процедуру DRAW_LINE и, следовательно, регистровая пара H'L' (альтернативные) также неизбежно коррумпируется.

Рисование окружности.

Здесь тоже ведется работа через встроенный калькулятор. Координаты центра окружности X и Y, а затем радиус R должны быть помещены на вершину стека калькулятора, а затем выполняется вход в процедуру CIRCLE через точку входа 232DH (9005) для изображения окружности. H'L'- коррумпируется.

Проверка точки экрана.

С помощью функции POINT (X,Y) Вы можете проверить включен или выключен пиксел экрана с координатами X, Y. Для имитации работы этой функции из машинного кода у Вас есть два пути. Во-первых, Вы можете прогрузить регистры B и C координатами X и Y, после чего вызвать процедуру POINT_SUB с точкой входа (22CEH = 8910), а во-вторых, можете установить X и Y на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 22CBH (8907). И в том и в другом случае результат будет оставлен на вершине стека. Снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733). Эта процедура перешлет результат в регистр A процессора. Если пиксел выключен, то есть имеет цвет PAPER, Вы получите 0, а если он включен, т. е. имеет цвет INK, Вы получите единицу.

Проверка атрибутов экрана.

С помощью функции ATTR (Y,X) Вы можете проверить установку атрибутов в заданном знакоместе с координатами Y, X. Здесь X -номер экранного ряда (1...24), а Y - номер столбца (1...32). Для имитации работы этой функции из машинного кода Вы можете загрузить регистры B и C координатами Y и X, после чего вызвать процедуру S_ATTR_S с точкой входа (2583H = 9603), а во-вторых, можете установить Y и X на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 2580H (9600). И в том, и в другом случае результат будет оставлен на вершине стека, снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733), которая перешлет результат в аккумулятор. Анализируя данные по битам, Вы сможете установить параметры цветовых атрибутов в заданном знакоместе.

Раскладку по битам см. на рис. 3.

Проверка содержимого заданного знакоместа.

В БЕЙСИКе вы можете воспользоваться функцией SCREEN$ (Y,X) для того, чтобы определить, есть ли какой-нибудь символ ASCII в данном знакоместе с координатами Y и X и если есть, то что это за символ. То есть, эта функция может исполнять сканирование экрана. Надо, правда, сказать, что работа этой функции в БЕЙСИКе не отличается надежностью. В работе активно используется стек калькулятора, на котором может образовываться такая путаница, что например IF STRING$ (0,0)=STRING$(0, 1) THEN PRINT "TRUE" будет всегда выдавать "TRUE" независимо от того, что содержится в знакоместах (0,0) и (0,1).

К счастью, мы можем организовать сканирование и из машинного кода, причем там такой путаницы не будет. Вы можете использовать процедуру ПЗУ S_SCRN$_S либо с точкой входа 2535H (9525), если Вы поместили Y и X на вершину стека калькулятора (именно в таком порядке), либо с точкой входа 2538H (9528), если вы поместили в регистр C номер экранного ряда, а в B - номер столбца.

И в том, и в другом случае результат будет оставлен на вершине стека калькулятора. Чтобы переслать его оттуда в регистры микропроцессора, воспользуйтесь процедурой FP_TO_AEDCB (2BF1H = 11249). После этого в BC будет содержаться 0, если такого символа в наборе ASCII не существует или 1, если символ опознан. Узнать, что это за символ, можно посмотрев содержимое аккумулятора - там будет содержаться его код.

Конечно, в результате сканирования смогут быть разысканы только символы ASCII - от 20H ("пробел") до 7FH ("копирайт"), т.к. только они имеются в таблице шрифта, на которую указывает системная переменная CHARS (23606 -5C36H), но это дело можно доработать. Мы не приводим здесь процедуру, которая способна просканировать весь экран и определить как символы ASCII, так и символы графики пользователя и символы блочной графики в связи с нехваткой места. Будущие читатели смогут познакомиться и с ней и с многими другими интересными и полезными вещами на страницах нашей новой книги.

СЛУЧАЙНАЯ ГРАФИКА

(Глава из книги "Дизайн Ваших программ")

Продолжая разговор о графике "Спектрума", мы представляем отрывок из готовящейся сейчас к изданию книги "Персональный компьютер СПЕКТРУМ. Дизайн ваших программ. " Это заключительный (четвертый) том готовящейся к выпуску серии, посвященной графике.

По мере окончания подготовки, книга будет предложена к изданию по регионам. Следите за нашей информацией.

СЛУЧАЙНАЯ ГРАФИКА

Есть один аспект, связанный с 48-килобайтными "Спектрумами", в котором проявляется ограниченность их ресурсов оперативной памяти - это графика. Здесь ограничения более, чем очевидны. Графические изображения могут чудовищно расходовать память Вашей машины.

Вам никогда не приходилось задумываться, почему одни программы имеют богатую графику, а другие - выглядят серыми и невыразительными? Что, может быть в первом случае работал хороший художник, а во втором - плохой? Да, конечно, для любительских программ это вполне справедливо, но если речь идет о крупной фирме, способной привлечь к работе и оплатить труд первоклассных дизайнеров, то ясно, что не в художнике дело. Он бы Вам изобразил все, что хотите, а вот как разместить все его идеи в ограниченном объеме памяти?!

Итак, графический дизайн программы оказывается тесно увязан с программными возможностями, с общей концепцией проекта, и насколько хороша будет программа, зависит в конечном итоге не только от художника, а от совместных усилий всей команды в целом, начиная с руководителя проекта и кончая самым последним программистом. Вот почему так важно еще на этапе предварительной проработки концепции будущей программы четко определиться с потребностями в графике и наметить для себя те программистские приемы и методы, которыми эти потребности будут обеспечиваться.

Сегодня мы Вас и познакомим с одним из таких приемов. Он состоит в том, чтобы предоставить генерацию графики самому компьютеру по некоторому заданному алгоритму, поставив тем самым производство графических изображений "на поток". У этого метода есть существенный недостаток, состоящий в том, что многие изображения могут оказаться "похожими" друг на друга и различаться лишь в деталях, но это уже вопрос баланса программы. Вы ведь можете использовать полученную таким образом графику только в качестве фоновой, накладывая на нее незначительные по размерам и по объемам расходуемой памяти графические объекты.

Такая техника применяется, как правило, в адвентюрных программах, поскольку именно в них приходится иметь дело с огромным количеством иллюстраций (например 32000 - в программах фирмы BEYOND - "Sorderon's Shadow" или, скажем, "Lords of Midnight", хотя есть и много более внушительные примеры). По всей видимости, первой применила такую технику фирма LEGEND в своей знаменитой программе "Valhalla".

Теперь давайте перейдем к конкретной задаче и посмотрим, как используя "спектрумовский" генератор случайных чисел, можно получить образцы такой "фоновой" графики (ее еще называют ландшафтной графикой) для Ваших программ.

Мы сразу должны оговориться, что полное исследование всех возможностей предлагаемого метода может занять у вас не один месяц и полностью осветить все варианты в рамках одной книги нет никакой возможности. Мы будем просто руководствоваться генеральным принципом "ИНФОРКОМа" - научить приемам и методикам, а скрупулезное исследование оставляем читателю на самостоятельную проработку.

Рассмотрим в качестве примера следующую Бейсик-программу. Не обращайте пока внимания на то, что она очень медленно работает, сейчас для понимания самой сути идеи нам это не так важно (Листинг 1).

ЛИСТИНГ 1.

```
1 DEF FN r(x)=INT(RND*x)
2 BORDER 0: INK 0: LET n=1: GO TO 1000
7 REM
8 REM ** Рисуем рамку и наносим фоновые цвета **
9 REM
10 CLS: PLOT 0,95: DRAW 128, 0: DRAW 0,80: PRINT AT 1, 17; "Рисунок номер..."; AT 3, 24; n;
    AT 0,0;
15 PRINT PAPER 5,',',',',' PAPER 1, ',' PAPER 6, ',',': RETURN
17 REM
18 REM ** Рисуем горы **
19 REM
20 INK 0:LET ht =FN r(24): LET mx=23: LET mn=0: LET p=FN r(2)
25 FOR i=0 TO 127: PLOT i,136: DRAW 0, 7+ht
30 IF p THEN LET ht=ht+FN r(2):IF ht>= mx THEN LET P=0: LET ht=mx: LET mn=1+FN r(7) : GO TO
    50
40 IF NOT p THEN LET ht=ht-FN r(2): IF ht<=mn THEN LET p=1: LET ht=mn:LET mx=8+FN r(16)
50 NEXT i: RETURN
77 REM
78 REM ** Рисуем озеро **
79 REM
80 INK 7: FOR i=0 TO 49
90 LET x = FN r(125): LET y=128+ FN r(8): PLOT x,y
100 DRAW FN r(4),0: NEXT i: RETURN
137 REM
138 REM ** Рисуем тростник **
139 REM
140 INK 4: FOR i=0 TO 127: LET mn=FN r(2): LET mx=1 +FN r (7)
150 PLOT i,119+mn: DRAW 0, mx-mn: NEXT i : RETURN
177 REM
178 REM ** Рисуем каменистую равнину **
179 REM
180 INK 0: FOR i=0 TO 49
190 LET x=FN r(128): LET y=104+FN r(8):PLOT x,y: NEXT i: RETURN
237 REM
238 REM ** Рисуем траву на переднем плане **
239 REM
240 INK 4: FOR i=0 TO 127: PLOT i,96: DRAW 0, FN r(8): NEXT i: RETURN
997 REM
998 REM ** Главный цикл рисования картинки **
999 REM
1000 CLS: RANDOMIZE n: GO SUB 10: REM ** Очистка картинки **
1005 GO SUB 20: REM ** Рисуем горы **
1010 GO SUB 80: REM ** Рисуем озеро **
1015 GO SUB 140: REM ** Рисуем ТРОСТНИК **
1020 GO SUB 160: REM ** Рисуем почву **
1025 GO SUB 240: REM ** рисуем траву **
1030 INK 0: PRINT #1; AT 1,9; FLASH 1; "Еще рисунок?"
1040 PAUSE 0: LET n=n+1: GO TO 1000
```

Как Вы видите, эта небольшая программа использует функцию генерации случайных чисел компьютера RND, с помощью которой случайным образом задаются параметры графических операторов БЕЙСИКа DRAW и PLOT. Программа довольно эффектно рисует пейзажи, содержащие горы, воду, почву и т. п. Но что самое интересное для программиста - так это то, что функция RND, как Вы должно быть знаете, вовсе не выдает случайные числа, а только лишь псевдо-случайные. То есть, имеется некоторая заранее предопределенная последовательность, из которой эти псевдослучайные числа и выбираются. Эта последовательность рассчитывается с помощью небольшой процедуры, имеющейся в ПЗУ

компьютера.

В качестве исходного параметра при работе этой процедуры используется значение системной переменной SEED, содержащейся в двух байтах по адресам 23670, 23671 (5C76H). Изменить содержимое этой системной переменной и начать генерацию новой псевдослучайной последовательности можно оператором RANDOMIZE n, что вы и видите в программе в строке 1000. В качестве параметра n может быть любое число от 0 до 65535, т.е. таким образом, Ваш генератор случайных чисел может выдать псевдослучайную последовательность из 65536-ти чисел, после чего они начнут повторяться (последовательность вырождается).

Если Вы зададите в операторе RANDOMIZE в качестве параметра n например номер картинки, то Вы всегда для одного и того же, номера будете иметь один и тот же пейзаж, но для разных n они будут различны.

Конечно, можно пойти еще дальше и, в зависимости от n, вызывать те или иные процедуры, которые внесут дополнительное разнообразие в Ваш рисунок, например, можно наложить какие-то небольшие рисунки на Ваш фоновый пейзаж.

Несложной заменой параметра цвета INK озеро, изображаемое на среднем плане, может быть преобразовано в болото, в снежный покров, в песчаную пустыню.

Проблемы скорости работы.

К сожалению, перед нами по-прежнему стоит проблема, связанная с медленной работой вышеприведенной программы. На первый взгляд, все это не так сложно, надо только заменить медленно работающие операторы PLOT, DRAW и RND на вызов из машинного кода тех процедур ПЗУ, которые им соответствуют, но результат будет ненамного лучше. Дело в том, что стандартная процедура ПЗУ, выполнявшая функцию RND, работает настолько медленно, что от того, что вы ускорите обращение к ней применением машинного кода, Вы мало чего достигнете. Надо искать другое решение.

Во-первых, честно говоря, нам и не нужна очень случайная последовательность для нашей задачи и нет смысла проходить все те манипуляции с числами, которые выполняет процедура ПЗУ.

Во-вторых, зачем нам 65536 случайных чисел? Может быть, на первый случай будет достаточно и 256-ти?

Ну, и если это все так, то мы сами можем создать где-то в памяти небольшую таблицу из 256 чисел, предварительно заполнить случайными числами, а затем брать их оттуда в готовом виде, не привлекая для этого обращение к ПЗУ.

Резервируем для этого область памяти, начиная с адреса 64744:

```
FOR i = 64744 TO 64799:
POKE i, INT(RND*255)): NEXT i
```

Теперь представьте, что у Вас есть некий указатель, который указывает на какой-то пункт в этой таблице. Предположим, вам понадобилось некоторое случайное число - Вы возьмете его из того пункта, на который указывает указатель, а сам указатель передвинете на следующую позицию для последующего использования. Когда указатель дойдет до конца таблицы, его совсем несложно опять перегнать в ее начало, так мы сможем обеспечить 256 различных картинок в своей программе, различающихся начальной позицией указателя. Конечно, при рисовании только одной картинки, указатель может много раз пробежать по таблице, но в этом нет никакой беды, потому что если например при рисовании травы и возникнет некое подобие регулярности, то оно не будет выглядеть неестественно, а графика, с помощью которой изображены горы, достаточно сложна, чтобы заметить в ней некоторые внутренние повторения. Для проведения математических или статистических исследований, такой подход был бы совершенно неприемлем, но для графики он вполне годится.

Трансляция БЕЙСИКа в машинный код.

Читатель, который интересуется только идеями, приемами и методами, мог бы на этом и завершить чтение данной главы и приступить к самостоятельным исследованиям по созданию образцов случайной машинной графики. Те же, кому необходим инструмент с

достойными рабочими характеристиками, получат ниже рекомендации о том, как значительно ускорить работу алгоритма, подключив в дело машинный код.

Прежде всего, для работы нам необходимы четыре процедуры:

- рисования точки (аналог PLOT);
- рисования линии (аналог DRAW);
- генерации случайных чисел (аналог RND);
- процедура для подготовки экрана (очистка экрана, рисование рамки, предварительное окрашивание фона в цвета PAPER) - назовем ее BCGRND от слова BackGround - фон, задний план.

PLOT.

С этой процедурой все просто. Ее даже не надо делать, вполне можно использовать процедуру, имеющуюся в ПЗУ по адресу 22E5H (8933), вызвав ее командой процессора CALL 22E5H.

Надо только предварительно перед вызовом этой процедуры выставить ее параметры в регистрах B и C микропроцессора. Так, если Вам надо выполнить PLOT x,y, то в регистр C должно быть загружено число "x", а в регистр B - число "y".

DRAW.

Здесь мы поступим аналогично, воспользовавшись процедурой ПЗУ, служащей для рисования линий и расположенной по адресу 24BAH (9403). Параметры "x" и "y" оператора DRAW x,y выставляются предварительно в регистрах C и B совершенно аналогично, но надо еще выставить в регистрах D и E знаки "y" и "x", соответственно. Во время своей работы, процедура "портит" содержимое регистровой пары HL регистров альтернативного набора. Поэтому, чтобы не было никаких коллизий с прочими процедурами, необходимо в общем случае запоминать содержимое этой пары на стеке процессора, а после окончания работы процедуры - восстанавливать его.

RND.

Работа этой процедуры представляет для нас несколько больший интерес. Нам надо, чтобы после ее вызова, она выдавала бы эквивалент того, что выдает оператор INT(RND*x), где "x" - число, предварительно загруженное в аккумулятор процессора. Результат работы процедуры при выходе из нее остается там же - в аккумуляторе. Одним словом, эта процедура должна делать то же самое, что и функция FN r(), определенная нами ранее в строке 1 вышеприведенной БЕЙСИК-программы.

Для работы этой процедуры Вам надо также предварительно заполнить 256-байтную таблицу псевдослучайных чисел, как это было показано выше. Сделайте это сами, а впоследствии мы сможем отгружать таблицу вместе с машинным кодом наших процедур единым блоком.

Обеспечивать задание начальной точки нашей псевдослучайной последовательности мы сможем, изменяя при помощи POKE значение переменной POINT, находящейся по адресу 65057. Так, заслав сюда номер Вашего рисунка, Вы получите его на экране.

BCGRND

Процедура служит для подготовки левой верхней четверти экрана к рисованию вашей картинки. Это оставляет еще достаточно места для текстовых сообщений, если Вы пишете адвентюрную или обучающую программу.

Процедура достаточно проста для тех, кто хоть немного знаком с машинным кодом. Для тех же, кто делает первые шаги, укажем только, что сначала она вызывает из ПЗУ процедуры PLOT и DRAW для рисования рамки, а затем приступает к закрашиванию фона цветами PAPER. Для неба - это 5 рядов цвета PAP_1 (в нашем случае это голубой), для озера - 2 ряда PAP_2 (синий), а для почвы – три ряда PAP_3 (желтый). Закрашивание выполняется командой процессора RST 16, служащей для вывода информации на экран, закрашивание рядов выполняется в цикле, окончание цикла задано командами процессора DJNZ.

Возможно, стоит пояснить, как происходит закрашивание каждого ряда, в принципе, чтобы окрасить ряд из 16-ти знакомест в цвет PAPER, надо напечатать 16 пробелов, но это расточительная операция. Здесь все сделано изящнее с помощью кода управления печатью

CHR 6 (строки 730, 830, 930). Этот управляющий символ называется PRINT COMMA - "Запятая Оператора PRINT". Его действие эквивалентно действию оператора TAB 16, т.е. он выводит на экран те самые нужные нам 16 пробелов.

Окончание заполнения ряда и переход к новому ряду выполняются печатью 13-го символа (CHR 13 = "ENTER") - строки 759, 850, 950.

Процедура организована таким образом, что Вы всегда имеете возможность поменять значения фоновых цветов PAP_1, PAP_2, PAP_3. Они заданы директивой АССЕМБЛЕРа DEFB в ячейках памяти 65137, 65138, 65139.

Распечатка этих четырех вспомогательных процедур приведена в Листинге_2. Наберите их с помощью любого доступного Вам АССЕМБЛЕРа и откомпилируйте в машинный код, начиная с адреса 65000 по 65139. Если Вы не забыли предварительно составить таблицу псевдослучайных чисел в адресах с 64744 по 64799, то можете выгрузить их совместно.

Листинг 2.

```
      10   ; Распечатка четырех
      20   ; вспомогательных процедур
      30   ; DRAW, PLOT, RND и BCGRND
65000 40              ORG 65000
      50   DRAW
      60   ; Процедура эквивалентна DRAW C, B
65000 70              EXX
65001 80              PUSH HL
65002 90              EXX
65003 100             LD DE,0101H
65006 110             CALL 24BAH
65009 120             EXX
65010 130             POP HL
65011 140             EXX
65012 150             RET
      160
      170
      180
      190  ; процедура эквивалентна PLOT C, B
      200  PLOT       EQU 22E5H
      210
      220
      230  RAND
      240  ; Процедура принимает из аккумулятора число
      250  ; и замещает его на результат вычисления
      260  ; INT (RND*A). Таблица случайных чисел в
      265  ; адресах 64741...64999 должна быть заполнена.
65013 270             LD (VALUE),A      ; Запомнили содержимое аккумулятора в ячейке
                                        ; с меткой VALUE.
65016 280             LD HL,64744       ; Начало таблицы случайных чисел.
65019 290             LD DE,(POINT)     ; Ввели текущее значение указателя.
65023 300             LD D,0
65023 310             ADD HL, DE        ; Встали на позицию указателя.
65026 330             LD A,E            ; Ввели значение указателя.
65027 330             INC A             ; Передвинули указатель,
65028 340             LD (POINT),A      ; Запомнили новое положение указателя.
65031 350             LD A, (HL)        , Приняли случайное число.
65032 360             CALL 2D28H        ; Эта процедура ПЗУ выполняет
                                        ; сразу две полезные функции -
                                        ; переводит число в действительную
                                        ; форму (с плавающей точкой)
                                        ; и помещает его на стек калькулятора.
65055 370             LD A,255H
65037 380             CALL 2D28H        ; Поместили на стек 255,
65040 390             RST 40            ; Включение калькулятора,
65041 400             DEFB 5            ; Команда калькулятора
                                        ; "деление". На стеке
```

```
                                  ; осталось "RND".
65042 410           DEFB 56       ; Выключение калькулятора.
65043 420           LD A,(VALUE)
66046 430           CALL 2D28H    ; Поместили на стек значе-
                                  ; ние VALUE.
65049 440           RST 40        ; Включение калькулятора,
65050 450           DEFB 4        ; Команда калькулятора
                                  ; "умножение". На стеке
                                  ; получено RND*VALUE.
65051 460           DEFB 39       ; Команда калькулятора INT.
65052 470           DEFB 56       ; Выключение калькулятора.
66053 480           CALL 2DD5H    ; очень важная процедура
                                  ; ПЗУ. Снимает число с вер-
                                  ; шины стека калькулятора и
                                  ; пересылает его в регистр A
                                  ; микропроцессора.
65050 490           RET
65057 500  POINT    DEFB 0        ; переменная POINT.
65058 510  VALUE    DEFB 0        ; Переменная VALUE.
      520
      530
      540  BCGRND
      550                         ; Процедура рисует рамку и фоновые полосы в
      560                         ; цвете PAPER, заданном в PAP_1, PAP_2, PAP_3.
65059 570           LD A, 2
65061 580           CALL 1601H    ; Вызовом этой процедуры
                                  ; выбирается устройство вывода
                                  ; информации. Поскольку
                                  ; в аккумуляторе установлено
                                  : число 2, вывод будет
                                  ; идти на экран.
65062 590           LD B,95
65064 600           LD C,0
65068 610           CALL PLOT
65071 620           LD B,0
65073 630           LD C,128
65075 640           CALL DRAW
65078 650           LD B,80
65080 660           LD C,0
65082 670           CALL DRAW
66085 680           LD A,17       ; Код PAPER.
65087 690           RST 16
65088 700           LD A,(PAP_1)  ; Установка цвета PAPER.
65091 710           RST 16
65092 720           LD B,5        ; 5 рядов - небо.
65094 730  LOOP1    LD A,6        ; Управляющий код CHR 6.
65096 740           RST 16
66097 750           LD A,13       ; Управляющий код ENTER.
65099 760           RST 16
66100 770           DJNZ LOOP1
65102 780           LD A,17
65104 790           RST 16
65105 800           LD A,(PAP_2)
65108 810           RST 16
65109 820           LD B,2        ; 2 ряда - озеро
65111 830  LOOP2    LD A,6
65113 840           RST 16
65114 850           LD A,13
66116 860           RST 16
65117 870           DJNZ LOOP2
65119 880           LD A,17
65521 890           RST 16
65122 900           LD A,(PAP_3)
65125 910           RST 16
65126 920           LD B,3        ; 3 ряда - земля
```

```
65128  LOOP3   LD A,6
65130          RST 16
65131          LD A,13
65133          RST 16
65134          DJNZ LOOP3
65136          RET
65137  PAP_1   DEFB 5        ; - голубой
65138  PAP_2   DEFB 1        ; - синий
65139  PAP_3   DEFB 6        ; - желтый
```

Изменив значение по адресу 65137 на 1, Вы вместо голубого неба получите синее и день превратится в ночь. А замена числа 1 на 4 по адресу 65138 превратит озеро в болото. Это ваши дополнительные резервы по созданию графики несложной по исполнению, малоемкой по расходуемой памяти, но способной внести своеобразную атмосферу в любую программу.

Теперь нам осталось воспользоваться созданными процедурами и довести дело до конца, написав программу, которая будет нам генерировать такие картинки в огромных количествах в доли секунды.

Распечатка представлена в Листинге_3. Он не снабжен подробными комментариями, поскольку в нем сделано все возможное, чтобы максимально следовать той БЕЙСИК-программе, с которой мы начинали. Она фактически и является комментарием этой процедуры.

Листинг_3.

```
65097          LD A,13
               ORG 65140

       MOUNT   ; Горы.
65140          CALL INIT
65143          LD A,24
65145          CALL RAND
65148          LD (HT),A
65151          LD A,23
65153          LD (MX),A
65156          XOR A
65157          LD (MN),A
65160          LD A,2
65162          CALL RAND
65165          LD (PP),A
65168  LOOP4   LD A,(II)
65171          LD C,A
65179          LD B,136
65174          CALL PLOT
65177          XOR A
65178          LD C,A
65179          LD A,(HT)
65182          ADD A,7
65184          LD B, A
65185          CALL DRAW
65188          LD A,(PP)
65191          CP 0
65193          JP NZ,UP
65196          LD A,2
65198          CALL RAND
65201          LD B,A
65802          LD A,(HT)
65205          SUB B
65206          LD (HT),A
65209          LD B,A
65310          LD A,(MN)
65213          CP B
65214          CALL NC,SWAP2
```

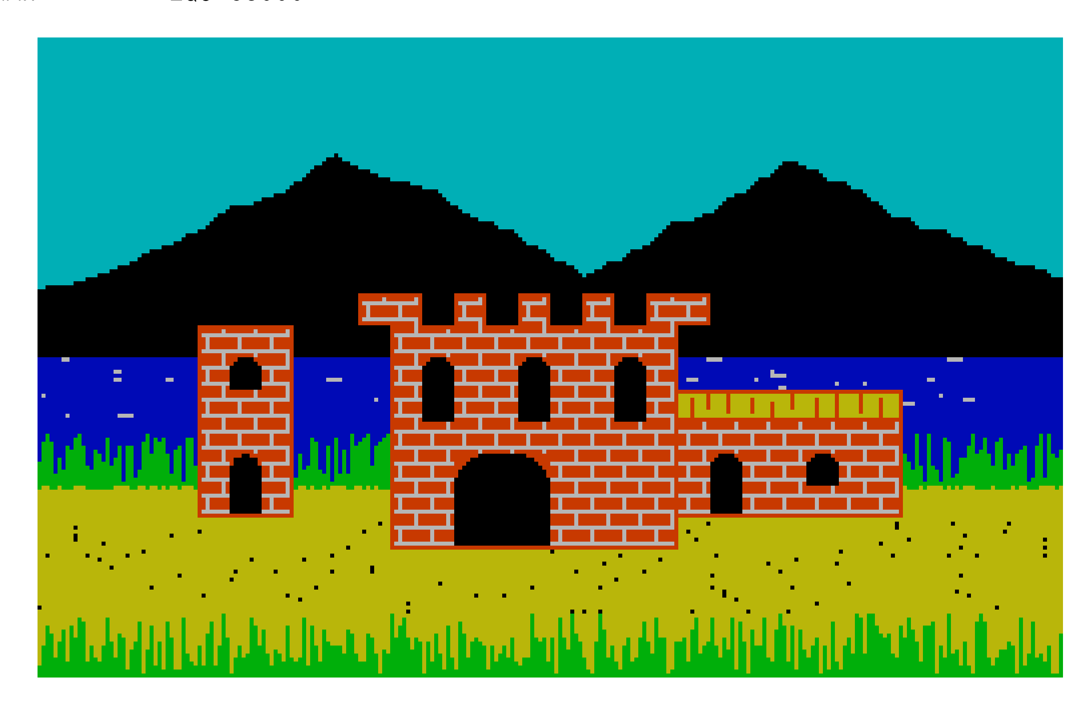
```
65217  1380  CONT       LD A,(II)
65220  1390             INC A
65221  1400             CP 128
65223  1410             RET Z
65224  1420             LD (II),A
65227  1430             JP LOOP4
65230  1440  UP         LD A,2
65232  1450             CALL RAND
65235  1460             LD B,A
65236  1470             LD A,(HT)
65239  1480             ADD A,B
65240  1490             LD (HT),A
65243  1500             LD A,(MX)
65246  1510             LD B,A
65247  1520             LD A,(HT)
65250  1530             CP B
65251  1540             CALL NC,SWAP1
65254  1550             JP CONT
65257  1560  SWAP1      LD A,B
65258  1570             LD (HT),A
65261  1580             LD A,7
65263  1590             CALL RAND
65266  1600             ADD A,1
65268  1610             LD (MN),A
65271  1620             XOR A
65272  1630             LD (PP),A
65275  1640             RET
65276  1650  SWAP2      LD (NT),A
65279  1660             LD A,16
65281  1670             CALL RAND
65284  1680             ADD A,8
65286  1690             LD (MX),A
65289  1700             LD A,1
65291  1710             LD (PP),A
65294  1720             RET
65295  1730  II         DEFB 0
65296  1740  PP         DEFB 0
65297  1750  HT         DEFB 0
65296  1760  MX         DEFB 0
65299  1770  MN         DEFB 0
       1780
       1790
       1800  LAKE       ; Озеро.
65300  1810             CALL INIT
65303  1620  LOOP5      LD A,125
65305  1630             CALL RAND
65308  1840             LD C,A
65309  1850             LD A,8
65311  1660             PUSH BC
65312  1870             CALL RAND
65315  1880             POP BC
65316  1890             ADD A,128
65318  1900             LD B,A
65319  1910             CALL PLOT
65322  1920             LD A,4
65324  1930             CALL RAND
65327  1940             LD C,A
65326  1950             LD B,0
65330  1960             CALL DRAW
65333  1970             LD A,(II)
65336  1980             INC A
65337  1990             CP 50
65339  2000             RET Z
65340  2010             LD (II),A
65343  2020             JP LOOP5
```

```
        2030
        2040
        2050  REEDS         ; Тростник
65346   2060                CALL INIT
65349   2070  LOOP6         LD A,2
65351   2080                CALL RAND
65354   2090                LD (MN),A
65357   2100                LD A,7
65359   2110                CALL RAND
65362   2180                ADD A,1
65364   2130                LD (MX),A
65367   2540                LD A,(II)
65370   2150                LD C,A
65371   2160                LD A, (MN)
65374   2170                ADD A,119
65376   2180                LD B,A
65377   2190                CALL PLOT
65380   2200                XOR A
65381   2210                LD C,A
65382   2220                LD A,(MN)
65385   2230                LD B,A
65366   2240                LD A,(MX)
65389   2250                SUB B
65390   2260                LD B,A
65391   2270                CALL DRAW
65394   2280                LD A,(II)
65397   2290                INC A
65398   2300                CP 128
65400   2310                RET Z
65401   2320                LD (II),A
65404   2330                JP LOOP6
        2340
        2350
        2360  GROUND        ; Земля.
65407   2370                CALL INIT
65410   3380  LOOP7         LD A,128
65412   2390                CALL RAND
65415   2400                LD C,A
65416   2410                LD A,8
65416   2420                PUSH BC
65419   2430                CALL RAND
65422   2440                POP BC
65423   2450                ADD A,104
65425   2460                LD B,A
65426   2470                CALL PLOT
65429   2480                LD A,(II)
65432   2490                INC A
65433   2500                CP 50
65435   2510                RET Z
65436   2520                LD (II),A
65439   2530                JP LOOP7
        2540
        2550
        2560  GRASS         ; Трава.
65442   2570                CALL INIT
65445   2580  LOOP8         LD A,(II)
65446   2590                LD C,A
65449   2600                LD B,96
65451   2610                CALL PLOT
65454   2620                LD A,8
65456   2630                CALL RAND
65459   2640                LD B,A
65460   2650                LD C,0
65462   2660                CALL DRAW
65465   2670                LD A,(II)
```

```
65468              INC A
65469              CP 128
65471              RET Z
65472              LD (II),A
65475              JP LOOP8


       INIT        ; процедура инициализации.
                   : Устанавливаем цвет PAPER
                   ; равный transparent -
                   ; то же самое, что
                   ; и PAPER 8.
65478              LD A,248
65480              LD (MASKT),A
                   ; Обнуление счетчика.
65483              XOR A
65484              LD (II),A
65487              RET

       MASKT       EQU 23696
       PLOT        EQU 8933
       DRAW        EQU 65000
```



Пример применения случайной графики.
Замок на переднем плане наложен более поздно.

После того, как Вы ассемблируете процедуры, приведенные в листингах, не забыв, конечно и о таблице случайных чисел, Вы принципе уже готовы к тому, чтобы начать рисовать. Дело осталось только совсем за малым: нужна программа-драйвер, которая объединит все части в единое целое и будет ими управлять. Ее можно сделать и на БЕЙСИКе (См. Листинг_4). Приведенный пример обеспечит Вам генерацию 256 различных картинок и прокрутку их одну за другой по нажатию клавиши.

Попробуйте, и Вы увидите, что скорость воспроизведения графики вполне достаточна для того, чтобы использовать ее в реальных программах и уж совсем несравнима с тем, что мы имели в БЕЙСИКе.

Практически мы с Вами затратили на генерацию 256-ти картинок около 800 байтов (вместе с таблицей случайных чисел), т.е. примерно по 3 байта на картинку. Это нормальное соотношение, но его можно было бы еще улучшить в несколько раз.

Во-первых, ясно, что если бы мы делали большее количество рисунков, то оно было бы лучше.

Во-вторых, хранение в памяти таблиц - это наиболее простой в смысле понятности способ и годится только для демонстрации вследствие своей расточительности. Профессионалы же пишут сами краткие и устойчивые алгоритмы генерации псевдослучайных последовательностей.

В третьих, если Вы проанализируете листинг 2, то увидите, насколько же мало байтов уходит на то, чтобы смоделировать тот или иной графический объект (горы, озеро, луг, грунт и т.п.). Пользуясь предложенной идеей, Вы всегда сможете развить число этих объектов (крепостная стена, изгородь, заросли кустарника, лес, море, скалы, звездное небо, космические тела и многое другое).

Ну и, наконец, в четвертых, мы уже говорили о том, что большое разнообразие при смысловой содержании и сохранении духа и атмосферы программы Вы сможете получить, накладывая на полученные пейзажи некрупные графические объекты, не занимающие большого места в памяти. Одним из рациональных приемов при этом является формирование их на основе блочной графики UDG.

ЛИСТИНГ_4.

```
1 REM ** БЕЙСИК-драйвер **
2 REM
3 BORDER 0: CLEAR 64743: LOAD ""CODE 65000
4 REM **Заполнение псевдослучайной таблицы**
5 REM
6 FOR i=64744 TO 64999: POKE i, INT(255*RND): NEXT i
7 LET n=0: REM **Инициализация счетчика**
8 REM **Главный цикл**
9 REM
10 LET n=n+1: POKE 65057, n
15 INK 0: CLS: RANDOMIZE USR 65059
20 PRINT AT 1, 17; "Номер рисунка..." AT 3,24; n
30 INK 0: RANDOMIZE USR 65140: REM **Горы**
35 INK 0: RANDOMIZE USR 65300: REM **Озеро**
40 INK 0: RANDOMIZE USR 65346: REM **Тростник**
45 INK 0: RANDOMIZE USR 65407; REM **Земля**
50 INK 0: RANDOMIZE USR 65442: REM **Трава**
60 PRINT #1; AT 1,9; FLASH 1: "Еще рисунок?": PAUSE 0
70 GO TO 10
```

Может быть, Вам потребуется не один набор UDG и Вы организуете эти наборы в банки. Впрочем, об этом речь еще впереди.

По мере готовности книги мы оповестим всех, кто готов заняться ее изданием и распространением в своих регионах.

Следите за нашей информацией!

КРИБЕДЖ

(Глава из книги "Настольные игры своими руками")

Вашему внимания предлагается глава из готовящейся к изданию книги "Персональный компьютер СПЕКТРУМ. Настольные игры своими руками. " Мы надеемся, что данный отрывок дает представление о содержании книги, которая будет предложена к изданию в ближайшее время.

КРИБЕДЖ

Считается общепризнанным тот факт, что крибедж - самая интересная карточная игра для двух играющих. Есть, правда, версии и для трех игроков и даже для четырех, но это уже не то. Вчетвером можно найти игру и поинтереснее (тот же бридж, например), а вот если Вас только двое, то крибедж может Вам очень и очень понравиться.

Предполагают, что изобрел крибедж и дал ему название английский поэт, лорд Джон Саклинг (1609-1642). Впоследствии первые колонисты завезли игру в Америку, где она и получила свой расцвет.

В чем прелесть крибеджа? Дело в том, что здесь, как и в любой другой карточной игре, определенную роль играет везение, но не слишком большую. У опытного игрока всегда есть возможность доказать, что счастье - это хорошо, но голова на плечах - лучше. Своим успехом крибедж во многом обязан правилам, которые сложными пожалуй не назовешь, но многообразными - можно. Счет очков идет настолько динамично, что для игроков даже существует специальное приспособление - доска с отверстиями. Игроки по мере набора очков переставляют свои колышки из отверстия в отверстие, стараясь как можно быстрее пройти путь к концу доски.

Предлагая Вам самостоятельно набрать эту программу, мы должны предварительно посвятить вас в правила этой игры, в правила подсчета очков и дать основы стратегии.

В игре используется полная колода карт - 52 листа. Старшинство карт следующее (сверху вниз): К,Д,В,10, ... 3,2,Т. Туз - младшая карта, каждая карта имеет свое достоинство. Все фигуры и десятка - по 10 очков, прочие - по номиналу, а туз - 1 очко.

Вся игра состоит из геймов.

Гейм состоит из раундов. Гейм закончен, когда кто-то наберет 121 очко. Победителю в гейме засчитывается одно очко, если его противник набрал более 61 очка и два очка, если тот набрал меньше. О том, до какого счета по геймам Вы будете играть - договоритесь сами, обычно играют до 6 или до 12-ти.

Сдача в игре производится по очереди, сдающий имеет значительное преимущество, он ведет активную игру. Его противник как бы держит оборону, сдача происходит в следующем порядке:

1. Каждому сдаются по 6 карт.

2. Каждый оставляет себе по 4 карты и по две карты сбрасывает на стол рубашкой вверх, не показывая их противнику. В результате у каждого осталось по 4 карты и на столе образовалась еще одна рука с четырьмя картами - эту руку называют КРИБ. Она принадлежит сдавшему.

3. После сноса криба противник "срезает" колоду и сдающий открывает верхнюю (после срезки карту). Эту карту называют "стартер". Она остается на колоде, но условно принадлежит всем играющим, в том числе и крибу, т.е. они имеют для игры по 4 карты, но для зачетов очков - по пять карт вместе со стартером.

Очки в раунде добываются двумя путями.

Во-первых, можно взять очки с игры, в этом смысле и сдающий и его противник равны между собой.

Во-вторых, даются очки за те комбинации карт, которые образовались на Вашей руке после сброса двух карт и вскрытия стартера. Поскольку сдающий кроме своей руки владеет

еще и крибом, то те комбинации, которые окажутся в нем после розыгрыша раунда, прибавятся к его очкам.

Игра.

Сначала рассмотрим, как набираются очки с игры. Игра производится поочередным выкладыванием карт на стол. При этом нельзя, чтобы сумма выложенных карт превысила 31 очко.

Первый выкладывает любую свою карту несдававший. Затем кладет карту сдатчик. При этом:

- если полученная сумма равна 15, ему запишется 2 очка;
- если его карта равна карте предыдущего хода, т.е. образуется "пара", ему запишется 2 очка.

Ход переходит к несдававшему. Он может сделать:

- "пятнадцать" (2 очка)
- "пару"(2 очка)
- "секанс" (три карты подряд, например В,10,Д или 3,2,Т - 3 очка).

Ход переходит и так далее.

В последующие ходы могут возникать и более сложные комбинации:

- секансы из более чем трех карт оцениваются во столько очков, сколько карт в секансе.
- "пэр рояль" - три одинаковые карты подряд дают 6 очков, т. к. из них можно составить 3 разные пары;
- "двойной пэр рояль" - 4 карты одного достоинства - 12 очков (6 разных пар).

Так игроки ходят по очереди до тех пор, пока один из них не сможет сделать ход. Ведь выходить за пределы 31 очка нельзя. Тогда он объявляет "ХОД" и его противник получает себе очко. Теперь противник может, если у него есть мелкие карты, выкладывать их на стол, оставаясь в пределах 31 очка. Так можно выложить за ход и две и три карты подряд. Если ему удается в результате своего хода набрать ровно тридцать одно, ему запишется 2 очка.

Когда и второй играющий не сможет сделать хода, то с оставшимися на руках картами игроки начинают новую выкладку, первым кладет карту тот, кто первым пропустил ход.

Ход последней картой (из восьми) дает еще одно очко, а если к тому же в результате этого хода образовалось 31, то добавляются 2 дополнительных очка.

Теперь, когда все карты выложены, игроки начинают подсчитывать свои очки за комбинации, которые у них были на руках к началу игры. Начинающим для простоты можно разобрать выложенные карты со стола по рукам и начать считать. В игре с компьютером подсчет сделает за вас машина, если Вы этого пожелаете. Стартер считается пятой картой для каждой из трех рук.

В зачет идут следующие комбинации:

"Пятнадцать" - 2 очка.

"Пара" - 2 очка.

"Пэр рояль" - 6 очков.

"Двойной пэр рояль" - 12 очков.

"Секанс" - по числу карт.

"Флешь" - 4 или 5 очков.

Флешь - это 4 карты одной масти в руке, но не в крибе (4 очка). Если ту же масть имеет стартер - то 5 очков, за 4 карты одной масти в крибе не дается ничего, но если эту масть имеет и стартер, то хозяин криба (сдающий) получает 5 очков.

"Его благородие" - Если у вас на руках есть валет той же масти, что и масть стартера, вы получаете одно очко.

"Челядь" - Если валет оказывается картой-стартером, то сдатчику засчитывается очко.

Теперь, рассмотрим некоторые примеры подсчета очков.

Комбинация К, Д, Д, В.

К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д - 2 очка.
Итого: 8 очков

Комбинация К, Д, Д, В, 10:
К, Д, В, 10 - 4 очка
К, Д, В, 10 - 4 очка
Д, Д, - 2 очка
Итого: 10 очков

Комбинация К, Д, Д, Д, В:
К, Д, В - 3 очка,
К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д, Д, - 6 очков
Итого: 15 очков

Комбинация К, Д, Д, В, В:
К, Д, В - 3 очка.
К, Д, В - 3 очка.
К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д, - 2 очка.
В, В, - 2 очка.
Итого 16 очков.

Комбинация 9, 8,8,7,7:
8,7 - 2 очка
8,7 - 2 очка
8, 7 - 2 очка
8, 7 -2 очка
7,7 - 2 очка
8, 8 - 2 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
Итого: 24 очка

Как видите, в крибедже не так просто правильно подсчитать все положенные Вам очки. В этом тоже есть интересный элемент игры. Считать нужно быстро и точно - этому вас научит правило МАГГИНЗ. Дело в том, что если Вы, объявляя свои очки, что-то забудете подсчитать, то Вам соперник имеет право Вас подправить, объявив "Маггинз!" и Ваши очки, которые он увидел, а Вы - нет, запишутся ему.

Вообще-то говоря, не принято при игре с новичками пользоваться этим правилом, поэтому в той программе, которая предлагается Вашему вниманию, есть возможность играть с этим правилом или без. Если Вы играете с ним, то сами должны ввести количество своих очков. Ошибетесь, компьютер сделает Вам "Маггинз!". Если Вы от этого правила откажетесь, то компьютер будет делать все расчеты и за Вас и за себя по всем правилам.

Стратегия в крибедже.

Исследование стратегий в крибедже можно разделить на два этапа. На первом этапе игроки должны определиться со сносом в криб. На первый взгляд, достаточно сосчитать все

очки в шести картах и отложить две с тем, чтобы сохранить максимальный счет в оставшихся четырех. Но иногда это приводит к тому, что приходится сносить в криб очки или ценные карты. Если криб принадлежит Вам, в этом нет ничего плохого, но если он принадлежит противнику, лучше ослабить руку и "надуть" криб. В криб противника очень опасно сносить пятерки, семерки, восьмерки и карты, которые могут стать основой для секанса. Лучшими для "надувания" являются очень высокие по достоинству, очень малые и карты "с разбросом", имеющие по достоинству промежуток в две и более карты.

Иногда приходится дробить комбинацию на руке, даже если криб принадлежит Вам. Как правило, лучше сохранить секанс и разбить пару, если это необходимо. Секанс в руке дает хорошие ожидания на поддержку от стартера, после того, как он будет открыт.

На втором этапе, в розыгрыше игры основной принцип стратегии состоит в том, чтобы не дать противнику объявить "пятнадцать" или "секанс". Самой надежной картой для первого хода является ход "четверкой", в ответ противник ни может сделать " пятнадцать", ни может помешать это сделать вам. Он, конечно, может дать "пару", но против пар защиты быть не может.

Десятиочковая карта (десятка или фигура) не очень хороший первый код, но он неплох, если у вас есть пятерка. Когда противник объявит "пятнадцать", Вы поставите "пару".

Гораздо приятнее ход, например семеркой. Если противник поставит "пару", Вы сможете тузом сделать "пятнадцать". Если противник поставит "пятнадцать" с помощью восьмерки, У Вас возможность дать "пару" и открытая и сверху и снизу секансная последовательность.

Одним словом, если у вас на руке есть "пятнадцать", например 9+6 то ходите старшей из них - это безопасно. При отсутствии других соображений ходить старшими более предпочтительно, сберегая младшие на конец раунда для игры под 31 очком.

Пример розыгрыша партии.

Рассмотрим в качестве примера розыгрыш одной партии, после чего перейдем непосредственно к нашей программе. В результате сдачи карт образовался следующий расклад:

СДАТЧИК:

ПК Б9 Т9 Б8 П7 ЧТ

ПРОТИВНИК

БВ П10 Ч8 Т7 Ч5 Т3

Противник сносит в криб 10-3.

Он не может сбросить 8-7 как прекрасную комбинацию (это "пятнадцать" и хорошая основа для "секанса"). Он не может сбросить 5, т.к. велика вероятность, что стартером окажется десятиочковая карта. Выбирая между бубновым валетом и пиковой десяткой, он предпочитает оставить валета, т.к. он может оказаться "его благородием".

У сдатчика уже есть двойной "секанс" 9-9-8-7, поэтому он легко сносит К-Т.

После переворота стартера оказалось, что это пиковая шестерка.

СТАРТЕР: П6

1. Противник заходит восьмеркой (можно было пойти и семеркой). - "Восемь".
2. Сдатчик делает ход семеркой. - "Пятнадцать" - "два очка".
3. Противник кладет семерку. - "Двадцать два" - "пара" - "два очка".
4. Сдатчик играет бубновой девяткой. - "Тридцать одно" - "два очка".
5. Противник открывает новый счет, он ходит валетом - "Десять".

6. Сдатчик может положить восьмерку или девятку, он кладет девятку, полагая, что девятки у противника нет, иначе тот на третьем ходу положил бы ее, делая "секанс", а он сделал "пару". - "Девятнадцать".

7. Противник играет пятеркой. - "Двадцать четыре".

8. У сдатчика восьмерка. Ее класть нельзя, т. к. сумма превысит 31 очко. Он объявляет "Ход!" и противник заносит себе очко.

9. У противника нет карт, счет открывается в третий раз. Ходит сдатчик, выкладывает свою восьмерку и получает очко "за последнюю карту".

Начинается подсчет комбинаций.

ПРОТИВНИК

"пятнадцать"	В-5	- 2 очка
"пятнадцать"	8-7	- 2 очка
"секанс"	8-7-6-5	- 4 очка
Итого		- 8 очков

СДАТЧИК

"пятнадцать"	8-7	- 2 очка
"пятнадцать"	9-6	- 2 очка
"пятнадцать"	9-6	- 2 очка
"пара"	9-9	- 2 очка
"секанс"	9-8-7-6	- 4 очка
"секанс"	9-8-7-6	- 4 очка
Итого		16 очков.

Сдатчик открывает криб, но там, к сожалению, нет ни одного очка.

Программа

Программа отличается значительным размером и представляет немалую сложность в отладке. Для упрощения этой трудоемкой работы мы привели в конце статьи комментарий к программе - он Вам поможет.

Программа была нами проверена уже после печати оригинал-макета и ошибки в нем крайне маловероятны, но опыт показал, что основную трудность представляет похожесть символов I, l и цифры 1. Поэтому в тех местах, где это особенно критично, мы даем пометку в строке REM.

На Ваше собственное усмотрение остается вопрос русификации Вашего компьютера. В вашем распоряжении память выше 60000, где вы можете разместить свой шрифт. О том, как это делается, читателям ZX-РЕВЮ, по-видимому, говорить не надо - мы об этом много раз писали. Можете этот адрес и изменить, скорректировав значение CLEAR в строке 9900.

Русификация с помощью символов UDG-графики здесь не проходит, т.к. они уже используются программой для изображения игральных карт и доски для крибеджа.

```
10 REM Здесь Вы введете команды,
20 REM необходимые для
30 REM русификации Вашего
40 REM компьютера
50 REM ************************
60 DEF FN t(x) = (x>9)*10+(x<10)*x
70 DEF FN s(x)=10*(x-INT (x))
80 GO TO 7000
500 REM ********************************* Выбор 2-х карт.
510 PRINT AT 7, 8; FLASH 1; "Я думаю"; AT 0,0;
565 FOR i=1 TO 5
570 FOR j=i+1 TO 6 : REM (I+1)
575 LET y=1: LET sum=0: LET f5=0
580 FOR x=1 TO 6
```

```
585 IF x=i OR x=j THEN GO TO 610
590 LET h(y)=c(x): LET i(y)=d(x) : LET h$(y)=c$(x): LET sum= sum+i(y)
595 IF i(y)=5 THEN LET f5=f5+1
600 LET y=y+1
610 NEXT x
620 GO SUB 1000
630 LET s=s+p+f+fl+f5: LET cr=0
640 IF c(j)=c(i) OR c(j)=c(i)+1 THEN LET cr=2
650 IF d(i)+d(j)=15 THEN LET cr=cr+2
652 IF debug THEN PRINT: FOR q=1 TO 4: PRINT h(q);" ";: NEXT q: PRINT , s;" ";cr;"   ";
655 IF dir=1 THEN LET s=s+cr: GO TO 665
660 LET s=s-cr
665 IF s>max THEN LET max=s: LET t(1)=i: LET t(2)=j: IF debug THEN PRINT "x";
670 NEXT j: NEXT i
675 LET y=0
680 LET x=t(1): GO SUB 5300
685 LET x=t(2): GO SUB 5300
690 PRINT AT 7,8;" ГОТОВ "
695 RETURN
1000 REM ********************************** Расчет руки
1070 LET p=0: LET f=0: LET fl=4 : REM (FL=4)
1110 IF sum=15 THEN LET f=2
1120 FOR x=1 TO c-1
1130 FOR y=x+1 TO c
1140 IF h(x)=h(y) THEN LET p=p+2
1150 IF i(x)+i(y)=15 THEN LET f=f+a
1155 IF c<5 THEN GO TO 1220
1160 LET t=0
1170 FOR z=1 TO c
1180 IF z=x OR z=y THEN GO TO 1200
1190 LET t=i(z)+t
1200 NEXT z
1210 IF t=15 THEN LET f=f+2
1220 NEXT y: NEXT x
1240 FOR x=1 TO x
1250 IF sum=1(x)=15 THEN LET f=f+2
1260 IF h$(x)<>h$(1) THEN LET f1=0: REM (FL=0)
1270 NEXT x
1360 IF c=4 THEN GO TO 1400
1290 IF sum-i(5)=15 THEN LET f=f+2
1300 IF fl=4 AND h$(5)=h$(1) THEN LET fl=5: REM (FL)
1310 IF fl=4 AND crib=2 THEN LET fl=0: REM Флешь в крибе может иметь не менее 5 карт.
1400 LET x=1: LET s=0
1410 IF x>3 THEN RETURN
1420 LET r=1: LET d=1
1430 IF h(x+1)=h(x)+1 THEN LET r=r+1: GO TO 1490
1440 IF h(x+1)>h(x) THEN GO TO 1470
1450 LET d=d+1: IF d<>3 THEN GO TO 1490
1455 IF h(x-1)<>h(x) THEN LET d=4
1460 GO TO 1490
1470 IF r>rlen THEN LET s=s+d*r : IF r=2 THEN LET s=s-d
1460 LET x=x+1: GO TO 1410
1490 LET x=x+1: IF x<c THEN GO TO 1430
1500 IF r>rlen THEN LET s=+d*r: IF r=2 THEN LET s=s-d
1510 RETURN
2000 REM ********************************** Выкладывание карт на стол
2005 LET p=0: LET s=0
2008 LET ct=FN t(c)
2010 IF sum+ct>31 THEN LET s=-1: RETURN
2055 LET n=n+1 : LET t(n)=c
2020 LET t=sum+ct: IF t=15 OR t=31 THEN LET s=2
2022 IF n=1 THEN RETURN
2025 IF ABS (t(n-1)-c)>= n THEN RETURN
2026 REM **Проверка на пару
2030 FOR x=n-1 TO 1 STEP 1
```

```
2040 IF t(n)<>l???(x) THEN GO TO 2100
2050 LET p=p+2
2060 NEXT x
2100 IF p=6 THEN LET p=12
2110 IF p=4 THEN LET p=6 2115 LET s=s+p
2120 IF p>0 THEN RETURN
2130 REM **Пар нет, проверим секансы
2200 IF n<3 THEN RETURN
2210 FOR I=3 TO n
2220 LFT y=1
2230 FOR x=n-I+1 TO n
2240 LET h(y)=t(x): LET У=У+1
2250 NEXT x
2270 GO SUB 2400
2330 LET r=l: REM (R=L)
2340 FOR x=1 TO I-1
2350 IF h(x)+1<>h(x+1) THEN GO TO 2360
2360 NEXT x
2370 IF r>p THEN LET p=r
2380 NEXT I
2385 LET s=s+p
2390 RETURN
2400 REM ******* Подпрограмма сортировки массива h() размерностью l;
2410 LET Z=0
2420 FOR x=1 TO l-1
2430 IF h(x)>h(x+1) THEN LET z=h(x): LET h(x)=h(x+1): LET h(x+1)=z
2440 NEXT X
2450 IF z<>0 THEN GO TO 2410
2460 RETURN
2700 REM ********* Ход игрока.
2710 IF nh = 4 THEN LET hgo=1: PRINT AT 6,9; "  ХОД!  ": RETURN
2715 LET m$= "Выберите карту": GO SUB 5500
2720 LET c=4: LET x=1: GO SUB 3700
2740 IF b(x)<>0 THEN GO TO 2765
2742 IF sum<22 THEN GO TO 2720
2745 PRINT AT 20, x*4-4; "ход?"
2747 LET m$= "Если нет хода, нажмите ЕНТЕР": GO SUB 5500
2750 LET hgo=x: GO SUB 3700
2755 PRINT AT 20, hgo*4-4; "    ": REM 4 пробела
2760 IF hgo<>x THEN GO TO 2740
2762 IF sum<safe THEN LET safe=sum
2763 RETURN
2765 LET k=x: BEEP .02,15
2770 LET c=a(k): GO SUB 2000
2775 IF s<0 THEN BEEP .2,20: LET m$="Сумма должна быть меньше 32": GO SUB 5500: GO TO 2720
2780 LET nh = nh+1
2785 LET sum=t: LET b(k)=0
2790 LET x = k: LET y=16
2800 GO SUB 5300
2820 LET x=nh: LET y=8
2830 LET x$=r$(c): LET y$=a$(k)
2840 GO SUB 5400
2650 LET player=man
2900 REM *** Подсчет очков
2910 PAPER 4
2920 PRINT AT 6,0; "Всего ";sum;"       ": REM 7 пробелов
2940 PRINT AT 6, 9; "   ";s; " "
2955 GO SUB 6000
2960 RETURN
3200 REM *** Ход компьютера
3220 IF nc =4 THEN LET cgo=1: RETURN
3225 LET m$="": GO SUB 5500
3230 IF debug=1 THEN PRINT #1; AT 0,0; n$; AT 0, 0;
3240 LET max=-9: LET x1=0
3250 FOR i=1 TO 4
```

```
3260 IF d(1)<0 THEN GO TO 3500: REM ход уже сделан
3270 LET c=c(i)
3280 GO SUB 2000: IF s<0 THEN GO TO 3500: REM не по правилам
3290 LET n=n-1: LET i(i)=s: REM Переиграть, счет сохранить
3300 REM специальные правила
3305 IF t+c=31 AND t<safe THEN LET s=s-1
3310 LET s=s+(t>15)-(t=21)+(t>=safe)-2*(t = 5)
3315 IF n>0 THEN GO TO 3400
3320 FOR j=1 TO 4
3330 IF i=j OR d(j)<0 THEN GO TO 3360
3340 IF t<>5 AND t+d(j)=15 THEN LET s=s+2
3350 IF ABS (c-c(j))<2 THEN LET s=s+2
3360 NEXT j
3380 GO TO 3450
3400 IF ABS (t(n)-c)>2 THEN GO TO 3450
3410 FOR j=1 TO 4
3420 IF j =i OR d(j) THEN GO TO 3440
3430 IF ABS (t(n)-c(j) )< = 2 THEN IF t+2*d(i)<32 THEN LET s = s+2
3440 NEXT j
3450 LET s=s+(RND)>.6)
3460 IF s>=max THEN LET max=s: LET x1=i
3490 IF debug THEN PRINT #1;c;"=":i(i);",";s;" ";
3500 NEXT i
3550 IF x1=0 THEN LET cgo=1: PRINT AT 6,9;" ХОД! ": RETURN
3560 LET c=c(x1): LET t=sum+FN t(c)
3570 LET n=n+1: LET t(n)=c
3580 LET sum=t: LET s=i(x1)
3590 LET nc=nc+1: LET d(x1)=-9
3600 LET x=nc: LET y=0
3610 LET x$=r$(c): LET y$=c$(x1)
3620 BEEP .02,12: GO SUB 5400
3630 LET player=zx
3640 GO TO 2900
3700 REM ********************************* Выбор карты
3710 PAPER 4
3720 PRINT AT 21,x*4-3; FLASH 1:
3725 IF INKEY$<>"" THEN GO TO 3725
3730 IF CODE INKEY$=13 THEN GO TO 3600
3750 IF INKEY$<>" " THEN GO TO 3730
3760 PRINT AT 21, x*4-3;" ";
3770 LET x=x+1: IF x>c THEN LET x=1
3790 GO TO 3720
3800 PRINT AT 21,x*4-3;" ";
3810 RETURN
4000 REM ********************************* Очередность ходов
4050 LET nh=0: LET nc=0
4065 LET safe=31
4080 GO SUB 4400
4090 IF dir<>zx THEN GO TO 4200
4100 REM ИГРОК
4110 IF done=1 THEN RETURN
4120 GO SUB 2700
4125 IF win>0 THEN RETURN
4130 IF sum=31 THEN GO SUB 4300: GO TO 4200
4140 IF cgo=0 THEN GO TO 4200
4150 IF hgo = 0 THEN GO TO 4100
4160 LET s=1: GO SUB 2900: IF win>0 THEN RETURN
4170 GO SUB 4300
4200 REM Компьютер
4205 IF done= 1 THEN RETURN
4210 GO SUB 3200
4220 IF win>0 THEN RETURN
4230 IF sum=31 THEN GO SUB 4300: GO TO 4100
4240 IF hgo=0 THEN GO TO 4100
4250 IF cgo=0 THEN GO TO 4200
```

```
4260 LET s=1: GO SUB 2900: IF win>0 THEN RETURN
4270 GO SUB 4300
4290 GO TO 4100
4300 REM ** Переворот карты
4310 LET y=0
4320 FOR x=1 TO nc
4325 GO SUB 5350
4330 NEXT x
4340 LET y=8
4350 FOR x=1 TO nh
4360 GO SUB 5350
4370 NEXT x
4400 REM **Следующий раунд
4405 LET done=0: LET s=0
4420 LET sum=0: LET n=0
4430 IF nh=4 AND nc=4 THEN LET done=1
4440 LET cgo=0: LET hgo=1
4450 GO SUB 2900: RETURN
4500 REM ********************************** Открытие и подсчет очков
4510 PRINT #1;AT 0,0;n$;
4515 LET c=5: LET rlen=2
4520 LET x=18: GO SUB 5600
4530 IF dir=zx THEN GO TO 4600
4540 FOR x=1 TO 5
4550 LET h(x)=c(x): LET h$(x)=c$(x)
4560 NEXT x
4570 LET up=1: LET c=4: GO SUB 5100
4575 LET m$="Считаю свои очки":GO SUB 5500
4580 LET player=zx: GO SUB 4700
4585 IF win THEN RETURN
4590 IF dir=zx THEN GO TO 4660
4600 LET m$="Считаю Ваши очки":GO SUB 5500
4605 FOR x=1 TO 5
4610 LET h(x)=a(x): LET h$(x)=a$(x)
4620 NEXT x
4625 LET c=4: LET y=11
4630 GO SUB 5000
4640 LET player=man: GO SUB 4700
4650 IF dir=zx THEN GO TO 4540
4660 LET m$="Готовы вскрыть криб? ": GO SUB 5500
4561 PAUSE 0
4665 LET x=18: GO SUB 5600
4670 GO SUB 5200
4675 LET m$="Считаю очки в крибе": GO SUB 5500
4660 GO SUB 4700
4690 RETURN
4700 REM ********************************** Подсчет очков
4705 LET nob=0: LET sum=0
4710 FOR x=1 TO 5
4715 IF x<5 AND h(x)=1 AND h$(x)=e$(5) THEN LET nob=1
4720 LET i(x)=FN t(h(x))
4735 LET sum=sum+i(x)
4740 NEXT x
4750 LET l=5: GO SUB 2400
4760 LET c=5: GO SUB 1000
4765 PRINT
4770 LET m$="": GO SUB 5500
4775 IF player=zx OR NOT mug THEN GO TO 4820
4780 INPUT "Введите свой счет. " ;ss
4785 IF ss<0 OR ss>50 THEN GO TO 4780
4790 IF ss=s+p+f+fl+nob THEN LET m$="Я согласен": LET s=ss: GO TO 4880
4795 LET m$=STR$ ss+" - неверно. Я получаю очки!"
4800 LET mug=2: LET player=zx
4820 PAPER 4
4830 IF f>0 THEN PRINT "15-ть -";f;"  ";
```

```
4840 IF p>0 THEN PRINT "пары-";p;
4845 PRINT
4850 IF fl>0 THEN PRINT "Флешь-";fl;"  ";
4860 IF s>0 THEN PRINT "секанс-";s;
4865 PRINT
4870 IF nob=1 THEN PRINT "очко - его благородие"
4875 LET s=s+p+f+fl+nob
4880 PRINT "Всего = ";s;
4885 IF mug = 2 THEN PRINT " - МОИ!"
4890 GO SUB 5500: GO SUB 6000
4891 IF mug=2 THEN LET player=man: LET mug=l: PAUSE 100
4895 RETURN
4900 REM ********************************** Снятие колоды - стартер выкладывается пятой
      картой в руке
4905 LET y=8: GO SUB 5350
4910 LET m$="Теперь я снимаю колоду"
4915 IF player=man THEN GO SUB 5500: PAUSE 50: GO TO 4930
4920 LET m$="Снимите колоду - любой клавишей": GO SUB 5500
4925 IF INKEY$="" THEN GO TO 4925
4930 LET r=RND*40+12.5
4935 LET k=INT p(r)
4940 IF k=s THEN GO TO 4930
4945 LET k$=s$(FN s(p(r)))
4950 LET x$=r$(k): LET y$=k$
4955 LET x=x-.25: LET y=7
4960 GO SUB 5400
4965 LET a(5)=k: LET a$(5)=k$
4970 LET c(5)=k: LET c$(5)=k$
4975 LET e(5)=k; LET e$(5)=k$
4980 RETURN
4985 LET e(5)=k: LET e$(5)=k$
5000 REM выкладка вашей руки
5020 FOR x=1 TO c
5030 LET x$=r$(a(x)): LET y$=a$(x)
5040 GO SUB 5400
5050 NEXT x: RETURN
5100 REM выкладка руки компьютера
5110 LET y=0
5120 FOR x=1 TO c
5130 LET x$=r$(c(x)): LET y$=c$(x)
5150 GO SUB 5400
5170 NEXT x: RETURN
5200 REM выкладка криба
5210 LET crib=1: LET y=11: IF dir=zx THEN LET y=0
5220 FOR x=1 TO 5
5230 LET h(x)=e(x): LET h$(x)=e$(x)
5240 LET x$=r$(e(x)): LET y$=e$(x)
5250 IF x<5 THEN GO SUB 5400
5270 NEXT x: RETURN
5300 REM стирание карты
5310 PAPER 4: GO TO 5370
5350 REM закрытая карта
5360 PAPER 2
5370 LET x$=" ": LET y$=x$: GO TO 5420
5400 REM открытая карта
5410 PAPER 7: IF y$=s$(1) OR y$=s$(3) THEN INK 2
5420 LET x1=4*x-4
5440 PRINT AT y,xl;x$;"  ";AT y+1,xl;y$;"  "
5450 PRINT AT y+2,x1;"    ";AT y+3,xl;"  ";y$
5460 PRINT AT y+4,x1;"  ";x$
5470 PAPER 4: INK 0: RETURN
5500 REM печать сообщений
5510 PRINT #1;AT l,0;n$;AT 1,0; m$
5520 RETURN
5600 REM Очистка экрана
```

```
5630 PAPER 4: PRINT AT 0,0;
5650 FOR y=1 TO 22: PRINT TAB x: NEXT y
5690 PRINT AT 0,0: RETURN
5700 REM Изображение доски для крибеджа
5710 PRINT AT 0,25; PAPER 6; "ВЫСП" :REM 3 пробела
5715 PRINT TAB 25; PAPER 6;"       " : REM 7 пробелов
5720 FOR y=1 TO 6 5730 PRINT TAB 25; PAPER 6; " A A " : REM UDG-символы.
5740 PRINT TAB 25; PAPER 6;" A A " : REM UDG-символы.
5750 PRINT TAB 25; PAPER 6; " B B " : REM UDG-символы.
5760 NEXT y
5765 PRINT TAB 25; PAPER 6; "       " REM: 7 пробелов
5770 PRINT n$; PAPER 6
5775 PRINT AT 2,28;"E";AT 19,28;"E" : REM UDG-символы.
5780 LET m$="КРИБЕДЖ"
5790 FOR x=1 TO 7
5800 PRINT AT 2*x+1,28;m$(x)
5810 NEXT x
5820 PRINT AT 1,0;: PAPER 4: RETURN
6000 REM корректировка счета
6010 BEEP .2,10
6015 IF s=0 THEN RETURN: REM вход в демонстрационную подпрограмму
6020 LET ss=v(player)
6030 IF ss>0 THEN GO SUB 6400
6040 LET v(player)=s(player)
6050 LET s(player)=s(player)+s
6060 IF s(player)>120 THEN LET win=player: GO TO 6085
6070 LET ss=s(player)
6060 GO SUB 6400
6085 PRINT PAPER 6; AT 20,25;s(man)
6090 PRINT PAPER 6; AT 20,31-(s(zx)>99)-(s(zx)>9):s(zx)
6095 RETURN
6400 LET x=25: LET v=1
6405 IF player=zx THEN LET x=31: LET v=3
6410 IF ss>60 THEN LET ss=ss-60
6415 IF ss<31 THEN LET ss=31-ss: GO TO 6440
6430 LET ss=ss-30: LET x=27: LET v=3: IF player=zx THEN LET x=29: LET v=1
6440 LET y=1+(ss+INT ((ss-1)/5))/2
6450 IF y<>INT y THEN LET y=INT y+1: LET v=v+1
6460 PRINT OVER 1; PAPER 6; AT y,x;v$(v)
6490 RETURN
7000 REM ********************************** Инициализация
7005 RANDOMIZE: LET debug=0
7010 BORDER 4: PAPER 4: INK 0: CLS
7020 LET x=0: LET y=0: LET z=0: LET i=0: LET j=0
7025 DIM h(8): DIM h$(6): DIM i(6)
7030 DIM a(6): DIM a$(6): DIM b(6): REM рука игрока
7035 DIM c(6): DIM c$(6): DIM d(6): REM рука компьютера
7040 DIM e(6): DIM e$(6): REM рука криба
7050 DIM g(2): DIM s(2): DIM v(2): DIM v$(5): REM счет
7055 DIM p(52): DIM t (12): REM колода
7060 INPUT "Добро пожаловать!" "Вам нужны инструкции? "; i$
7070 IF i$(1):"y" OR i$(1)="Y" THEN GO SUB 9000: GO TO 7080
7075 PRINT #1; "Пожалуйста немного подождите. "
7080 FOR x=1 TO 13: READ k$
7085 FOR y=0 TO 7: READ z
7090 POKE USR k$+y,z
7095 NEXT y: NEXT x
7300 LET zx=1: LET man=2
7305 REM ***Графика пользователя
7310 LET v$="GFJIK"
7315 LET s$="HCDS"
7320 LET r$="T23456739RВДК": REM здесь R - символ UDG-графики.
7330 LET n$="                                ": REM 32 пробела
7335 LET o$-"Нажмите любую клавишу"
7340 IF i$="y" THEN GO SUB 9050
```

```
7350 REM: подготовка колоды
7355 LET z=0
7360 FOR x=1 TO 13
7365 FOR y=. 1 TO .4 STEP .1
7370 LET z=z+l: LET p(z)=x+y
7375 NEXT y 7380 NEXT x
7390 LET g(man)=0: LET g(zx)=0
7400 REM ********************************* Начало новой игры
7404 LET mug=0
7405 INPUT "Играем с правилом МАГИНЗ? ";m$: IF m$="y" OR m$="Y" THEN LET mug=1
7406 GO SUB 5700
7410 PRINT #1;AT 0,0; "Младшая карта сдает. "
7420 LET s(zx)=0: LET s(man)=0: LET v(1)=0: LET v(2)=0
7425 LET dir=zx: LET s= 0: LET player=zx: LET win=0
7430 LET x=2: GO SUB 4900: LET s=k
7435 PAUSE 50
7440 LET player=man: LET x=5: GO SUB 4900
7450 IF s<k THEN LET dir=man
7470 PAUSE 50: LET x=23: GO SUB 5600
7500 REM ************************** Тасование колоды и сдача
7510 LET m$=" Я сдаю"
7520 IF dir=man THEN LET m$="Вы сдаете"
7530 PRINT #0;AT 0, 0; m$; n$
7535 LET m$= "Тасую колоду"
7540 GO SUB 5500
7610 FOR x=1 TO 51
7620 LET y=INT (RND*(53-x))+x: LET z=p(x); LET p(x)=p(y): LET p(y)=z
7630 NEXT x
7635 LET m$="": GO SUB 5500
7640 REM **Сортировка карт
7650 FOR y=1 TO 7 STEP 6
7655 LET z=0
7660 FOR x=y TO y+4
7670 IF p(x)>p(x+1) THEN LET z=x: LET j=p(x): LET p(x)=p(z+1):LET p(x+1)=j
7680 NEXT x: IF z>0 THEN GO TO 7655
7690 NEXT y
7700 REM ** сдача
7710 LET i=0: LET j=16
7720 IF dir=zx THEN LET i=16: LET j=0
7730 FOR x=1 TO 6
7740 LET y=1: GO SUB 5350
7750 LET c(x)=INT p(x): LET c$(x)=s$(FN s(p(x)))
7760 LET d(x)=FN t(c(x))
7770 LET y=j: GO SUB 5350
7780 LET a(x)=INT p(x+6): LET a$(x)=s*(FN s(p(x+6)))
7790 LET b(x)=1
7795 NEXT x
7799 REM********************************* Снос карт в криб
7800 LET c=6: LET y=16: GO SUB 5000
7805 IF debug THEN GO SUB 5100
7810 LET m$=" Подождите, пока я снесу 2 карты.": GO SUB 5500
7820 LET c=4: LET rlen=l: LET crib=0
7825 LET max=-99: LET t(1)=1: LET t(2)=6
7830 GO SUB 500
7840 LET m$="Выберите 2 карты": GO SUB 5500
7845 IF i$="y" THEN GO SUB 9300: GO SUB 9400
7850 LET c=6: LET y=16: LET x=1
7860 GO SUB 3700: GO SUB 5350
7870 LET t(3)=x: GO SUB 3700
7880 LET x$=r$(a(x)): LET y$=a$(x)
7885 IF x=t(3) THEN GO SUB 5350: GO TO 7860
7890 LET t(4)=x: GO SUB 5350
7895 REM** Закрытие рук
7900 FOR x=1 TO 2
7905 LET e(x)=c(t(x)): LET e$(x)=c$(t(x))
```

```
7910 LET c(t(x))=0
7915 LET e(x+2)=a(t(x+2)): LET e$(x+2)=a$(t(x+2))
7920 LET a(t(x+2))=0
7925 NEXT x
7930 LET y=1: LET z=1
7935 FOR x=1 TO 6
7940 IF c(x)=0 THEN GO TO 7955
7945 LET c(y)=c(x): LET d(y)=d(x)
7950 LET c$(y)=c$(x): LET y=y+1
7955 IF a(x)=0 THEN GO TO 7970
7960 LET a(z)=a(x): LET a$(z)=a$(x) : LET z=z+1
7970 NEXT x
7980 LET x=24: GO SUB 5600
7985 IF i$="y" THEN GO SUB 9450
7990 PRINT #0; AT 0,0; n$
7999 REM **** Ход картой
8000 LET s=0: LET player=dir
8010 LET x=6: GO SUB 4900
8020 IF k=11 THEN LET m$= "2 очка за его челядь": GO SUB 5500: LET s=2: GO SUB 6000: PAUSE
  40
8030 LET c=4
8040 LET y=16: GO SUB 5000
8050 GO SUB 4000
8060 IF win THEN GO TO 8300
8065 IF i$="y" THEN GO SUB 9200
8070 GO SUB 4500
8080 IF win THEN GO TO 8200
8100 LET m$="Готовы к следующему ходу ?": GO SUB 5500
8105 IF INKEY$="" THEN GO TO 8105
8110 LET dir = dir+1: IF dir>2 THEN LET dir=1
8120 LET x=25: GO SUB 5600
8130 IF i$<>"y" THEN GO TO 7500
8140 INPUT "Продолжать инструкции? " ; i$
8150 IF i$="Y" THEN LET i$="y"
8160 GO TO 7500
8200 REM ***Победитель
8210 LET m$= "Примите поздравления с победой!"
8220 IF win=zx THEN LET m$="Вам не повезло - моя победа!"
8230 PRINT #1;AT 0,0;m$
8240 PRINT PAPER 6; AT 18,28; FLASH 1;v$(5)
8250 FOR x=1 TO 60 STEP 2: BEEP .02,x: NEXT x
8260 LET g(player)=g(player)+1 8265 LET m$=o$: GO SUB 5500
8270 PAUSE 0
8280 LET x=23: GO SUB 5600
8290 PRINT AT 4,8; "Счет: ": AT 5,7
8300 PRINT AT 7,8; "Вы   ";g(man)
8310 PRINT AT 9,8; "Комп.";g(zx)
8320 INPUT "Сыграем еще? "; m$
8330 IF m$="n" OR m$="N" THEN STOP
8380 GO TO 7400
9000 REM ******************** Инструкции
9005 LET i$="y"
9010 CLS : PRINT "  Крибедж - игра для двух игро-
     ков. Каждый получает до 6
     карт, из которых должен 2
     снести, образуя третью руку,
     называемую крибом. Криб счи-
     тается в пользу сдатчика.
     Из колоды вскрывается карта,
     принадлежащая одновременно
     всем трем рукам - стартер."
9020 PRINT ' "    Тузы оцениваются в одно
     очко. Все фигуры - в 10 очков."
```

```
9030 PRINT : PRINT "   Во время игры Вы увидите, как
        за некоторые комбинации карт
        начисляются очки."
9040 RETURN
9060 LET m$=o$: GO SUB 5500
9065 PAUSE 0
9070 CLS: GO SUB 5700
9075 PRINT "    Полученные очки отмечаются
        на игровой доске перемещением
        колышков. Победит тот, кто
        первым наберет 121 очко (два
        жды обойдет доску)."
9090 LET m$="Для демонстрации нажмите пробел." : GO SUB 5500
9095 PAUSE 0: IF INKEY$<>" " THEN GO TO 9175
9100 LET s(man)=0: LET s(zx)=0
9105 LET m$="Посмотрим мои очки": GO SUB 5500
9110 FOR i=zx TO man
9115 LET player=i: LET win=0
9120 LET s(player)=0: LET v(player)=0
9130 LET s=INT (RND*10): GO SUB 6015
9140 IF win=0 THEN GO TO 9130
9145 PRINT PAPER 6; AT 18,28;v$(5)
9150 LET m$="Посмотрим Ваши очки":GO SUB 5500
9155 NEXT i
9160 LET m$ "Для повтора нажмите ПРОБЕЛ": GO SUB 5500: GO TO 9095
9180 PRINT AT 9,0;
      "Когда Вам надо выбрать " '
      "карту, перемещайте ука-" '
      "затель клавишей ПРОБЕЛ " '
      "и делайте свой выбор - " '
      "клавишей ЕНТЕР.        "
9190 PRINT AT 9,0;
      "Если Вы будете играть  " '
      "с правилом МАГГИНЗ, то " '
      "должны сами считать    " '
      "свои очки. Я отберу их," '
      "если Вы ошибетесь.     "
9195 RETURN
9200 LET x=25: GO SUB 5600
9210 PRINT
      "Каждый сам считает свои" '
      "очки                   "
9220 PRINT
      "Затем сдававший счита- " '
      ет криб.                "
9240 GO SUB 9305: GO SUB 9400
9245 LET m$=o$: GO SUB 5500
9250 PAUSE 0: LET x=25: GO SUB 5600
9255 LET x=6: LET y=8: GO SUB 5350
9260 LET x=x-.25: LET y=7: LET x$-r$(c(5)): LET y$=c$(5)
9270 GO SUB 5400
9280 RETURN
9300 PRINT AT 0,0;
9305 PRINT
      "Очки начисляется за    " '
      "комбинации:            " '
9310 PRINT "За 15 баллов......2 очка"
9320 PRINT "За пару...........2 очка"
9330 PRINT "За три...........6 очков"
9310 PRINT "За четыре.......12 очков"
9350 PRINT "За секанс..очко за карту"
9360 RETURN
9400 PRINT "Флешь из четырех..4 очка"
9410 PRINT "Флешь из пяти....5 очков"
9420 PRINT "Валет масти стартера...."
```

```
9430 PRINT "................ 1 очко"
9440 RETURN
9450 PRINT AT 0,0;
     "Мы ходим по очереди,   " '
     "пока сумма очков не    " '
     "приблизится к 31.      "
9460 PRINT
     "Набравший 31 балл, по- " '
     "лучает 2 очка. Ближай- " '
     "ший к 31 получает очко."
9470 PRINT
     "Комбинации на руках да-" '
     "ют очки:               "
9510 GO SUB 9320
9520 PRINT
     "Если вам нечем ходить, "
     "выбирайте пустую карту "
9530 LET m$=o$: GO SUB 5500
9540 PAUSE 0: LET x=25: GO SUB 5600
9590 RETURN
9600 DATA "a"
9610 DATA 0,195,195,0,0,195,195,0
9620 DATA "b"
9630 DATA 0,195,195,0,0,0,0,0
9640 DATA "c"
9650 DATA 56,56,254,254,214,16,1 6,56
9660 DATA "d"
9670 DATA 16,56,124,254,254,124,56,16
9680 DATA "e"
9690 DATA 0,24,24,0,0,0,0,0
9700 DATA "f"
9710 DATA 240,255,240,0,0,0,0,0
9720 DATA "g"
9730 DATA 0,0,0,0,240,295,240,0
9740 DATA "h"
9750 DATA 66,238,254,124,124,56,16,16
9760 DATA "i"
9770 DATA 15,255,15,0,0,0,0,0
9760 DATA "j"
9790 DATA 0,0,0,0,15,255,15,0
9800 DATA "k"
9810 DATA 28,28,28,28,8,8,8,8
9820 DATA "s"
9B30 DATA 16,56,124,254,254,146,16,56
9840 DATA "r"
9850 DATA 0,76,210,82,82,82,76,0
9900 CLEAR 59999
9999 PAPER 7: BORDER 7: INK 0: CLS: LIST
```

Комментарий

Графика пользователя используется в строках: 5730, 5740, 5750, 5775,7310,7315 и 7320.

Стратегия работы программы состоит из двух фаз. Первая фаза - когда компьютер решает, какие же 2 карты ему следует снести в криб. Здесь программа просматривает все комбинации из шести карт по 4 и каждый раз рассчитывает силу руки. Оценивается также сила двух сносимых в криб карт. Этих комбинаций всего 15 и они занимают около 1 секунды. Всего расчет оптимального сноса занимает порядка 20 секунд. Это самая медленная часть программы, но если Вы попробуете с ней сыграть, то увидите, что это не намного дольше, чем ожидание решения от живого партнера, так что здесь со скоростью работы проблемы особой нет.

Вторая фаза - розыгрыш рук и ведение счета. Здесь программа рассчитывает итог,

который может быть получен при том или ином ходе и применяет некоторый эвристический подход, который в определенной степени может давать и непредсказуемые результаты. Те, кому интересно посмотреть, как это происходит, могут в строке 7005 задать для переменной debug значение LET debug=1.

Несмотря на то, что программа полностью написана на БЕЙСИКе, в ней многое сделано для структурирования. Наиболее часто использующиеся подпрограммы, вынесены в начало для минимизации времени доступа к ним. Ниже мы рассмотрим назначение основных подпрограмм.

Данные также структурированы.

Колода представлена массивом из пятидесяти двух чисел - p(). Целая часть каждого числа выражает собой достоинство карты (1...13), а дробная часть - ее масть (0.1...0.4). Массив организуется случайный образом во время тасования колоды.

Другие важнейшие массивы:
Партнер:
a()- достоинство карт;
a$()-масти карт;
b()-вспомогательный массив.

Компьютер:
c() - достоинство карт;
c$() - масти карт;
d()- вспомогательный массив.

Криб:
e()- достоинство карт;
e$()-масти карт;

Вычислительные массивы:
h()- достоинство карт:
h$()-масти карт;
i()-вспомогательный массив.
t()-массив, содержащий карты, выложенные на стол во время ходов. Карты стола.
s() -счет.

Все вычисления, связанные со счетом, производятся после копирования массивов того или иного игрока в вычислительные массивы. Использование прочих переменных мы рассмотрим в составе основных подпрограмм.

500. Подпрограмма рассматривает все возможные комбинации 4-х карт из шести с целью определения что же сносить в криб. Сами расчеты делает другая подпрограмма (1000), к которой происходит обращение по мере необходимости.

На входе в подпрограмму задаются массивы, описывающие руку компьютера - c(), c$(),d().

На выходе она выдает t(1) и t(2) - позиции тех карт, которые должны быть снесены в криб.

1000. Подпрограмма рассчитывает очки, имеющиеся на руке. Она вызывается не только при расчете сноса, но и при проведении расчетов после открытия карт. Параметр c указывает сколько карт принимаются в расчет. При расчете сноса c=4, при подсчете итогов c=5 (включая карту-стартер).

Другие переменные:
rlen - содержит минимальную длину секанса.
sum - суммарная сила карт руки.

h(),i(),h$() - временно содержат данные той руки, для которой идет подсчет очков.
crib=1, если расчет идет для криба (флаг криба).
На выходе эта подпрограмма выдает:
p - очки за "пары";
f - очки за комбинации "пятнадцать";
s - очки за секансы;
fl - очки за флеши.

2000. Подпрограмма переносит карту из массива игрока в массив стола (делается ход, если он законный). Изменяется сила руки игрока и общий счет. На входе подпрограмма получает:
c - номер карты, которой делается ход;
sum - текущая сила руки;
t() - массив карт уже лежащих на столе.
n - количество карт, лежащих на столе.
На выходе подпрограмма выдает:
s - текущий счет (если сделан был незаконный ход, то этот параметр выдается отрицательным);
t() и n - измененные значения.

2400. Подпрограмма выполняет сортировку поступившего массива h(), имеющего длину l (буква L).

2700. Подпрограмма обслуживает ход игрока.
Вход:
nh количество карт сыгранных с этой руки;
a(), a$ () - карты, имеющиеся на руке;
b() - содержит нули для карт, которые уже сыграны.
Выход:
hgo=1, если ни один ход не может быть сделан;
win=man, если игрок достиг 121 очка и победил.

2900. Подсчитывает текущую сумму очков, корректирует счет, если необходимо.
Вход:
sum - текущая сумма очков;
s - счет;
player - указывает на игрока.
player=zx - компьютер;
player=man - человек.
Выход: нет.

3200. Рассчитывает оптимальный ход с руки компьютера. Перебираются и оцениваются все возможные ходы.
Вход:
nc - количество карт сыгранных с этой руки;
c(), c$ - карты, имеющиеся на руке;
d () - содержит нули для карт, которые уже сыграны.
Выход:
cgo=1 , если ни один ход не может быть сделан;

3700. подпрограмма обеспечивает игроку выбор карты для хода путем перемещения указателя. SPACE – перемещение, ENTER - ход.
Вход:
c - количество карт на руке.

Выход:
x - номер избранной карты.

4000. Подпрограмма последовательно вызывает подпрограммы, отвечавшие за ход игрока и ход компьютера, Подпрограмма в строке 3400 используется для того, чтобы перевернуть карты на столе, когда текущий счет дойдет до 31. Вызов подпрограммы 4400 сбрасывает текущий счет на ноль.
Вход:
dir - содержит сведения о том, кто сдавал карты.
dir=zx - компьютер;
dir=man - человек.
Выход:
done=1, если ход сделав;
win - указывает на победителя;
win=zx - компьютер;
win=man - человек.

4500. Изображает на экране раскрытые карты игроков и результат подсчета очков.

4700. Печатает счет для руки. Если в силе правило "МАГГИНЗ", предлагает игроку самому ввести свой результат.
Вход:
mug=1, если действует "МАГГИНЗ";
h(),h$() - карты анализируемой руки.

4900. Подпрограмма выполняет "срезку" колоды для вскрытия стартера.
Вход:
x - позиция указателя в колоде.
Выход:
k - достоинство стартера;
k$ - его масть.

5000. Изображает на экране карты игрока.
y - вертикальная позиция на экране;
c - количество карт в руке.

5100. изображает на экране карты компьютера.
c - количество карт в руке.
up= 1, если карты следует положить в открытом виде.

5200. Изображает на экране карты криба и копирует их из e() в h() для подготовки к расчетам.

5300. Стирает карты с экрана.
x - позиция карты от 1 до 6.
y - позиция экрана по вертикали.

5350. Печатает карту, изображенную рубашкой вверх (в темную).
x - позиция карты от 1 до 6.
y - позиция экрана по вертикали.

5400. Печатает открытую карту.
x - позиция карты от 1 до 6
y - позиция экрана по вертикали.

x$ - достоинство карты;
y$ - масть.

5500. Печатает в нижней части экрана текст сообщения из переменной m$.

5600. Очищает левую часть экрана.

5700. Рисует доску для крибеджа.

6000. Корректирует счет на доске для крибеджа.

7000. Инициализация программы. Объявление массивов, задание графики пользователя, составление колоды.

7400. Начало игры. Сбрасывается счет, вытягиванием младшей карты определяется, кто будет сдавать первый.

7500. Тасуется и сдается колода.

7800. Сброс карт, вызывается подпрограмма 500 для определения карт сноса. Дважды вызывается подпрограмма 3700 для сброса карт игрока. Массивы рук перерабатываются для удаления образовавшихся зазоров.

8000. Начало розыгрыша раунда. Срезается колода вызовом 4900. Затем игроки по очереди ходят (4000) и карты открываются для подсчета очков (4500).

8200. Определяется победитель. Изменяется счет в геймах, начинается новый гейм.

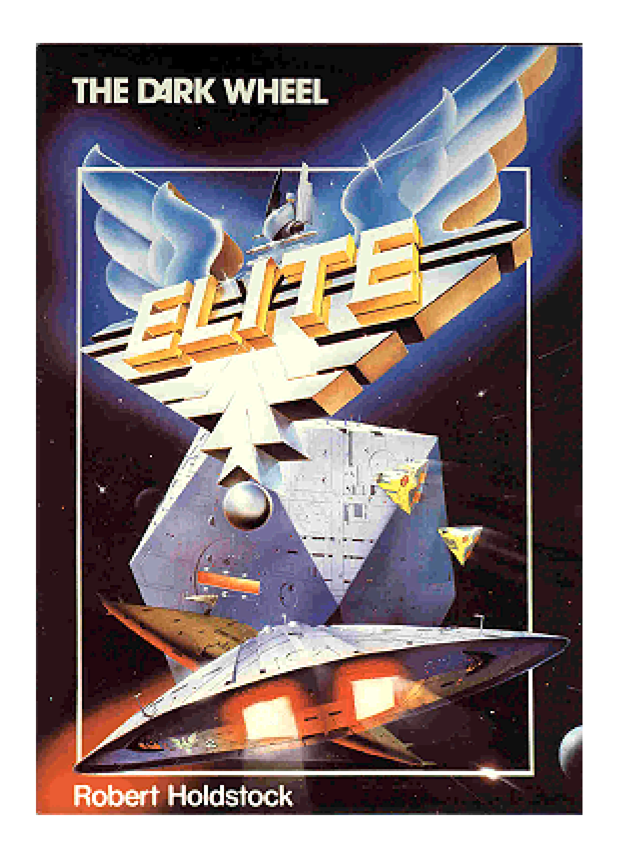
9000, Печать инструкции по игре. Переменная i$ определяет, давать инструкции или нет.

9600. Данные по UDG-графике.

9990. Инициализация цвета, установка верхней границы области БЕЙСИКа оператором CLEAR.

В заключение мы желаем Вам успеха в работе с этой интересной программой, надеемся, что у Вас достанет мужества и терпения, чтобы ее набрать и отладить. Ждем Вас в дальнейшем на страницах этой новой готовящейся книги.

Выполняя свое обещание, данное многочисленным поклонникам программы ELITE, сегодня мы начинаем печать повести THE DARK WHEEL, написанной Робертом Холдстоком по мотивам программы. Для нас это совершенно необычный опыт, т.к. в жанре художественного перевода мы, мягко говоря, себя никогда не пробовали. Что ж посмотрим, как будет выглядеть этот первый блин.



THE DARK WHEEL

ГЛАВА 1.

Торговый корабль "Авалония" плавно отошел от места орбитальной стоянки над планетой Лейв и начал маневр, приближаясь к исходной точке гиперперехода. Восемнадцать минут - ровно столько оставалось жить как кораблю, так и одному из двух членов ее экипажа.

Орбитальная станция отошла в тень, включились двигатели и, содрогаясь от вибрации, маленький корабль пошел к последнему гиперпрыжку. Внизу неспешно вращался Лейв во всем своем зелено-голубом великолепии. Шесть завитков пурпурных и белых облаков несли штормы морям, ливни континентам и обещали несколько влажных дней густым лесам и глубоким ущельям. Как яркие стеклянные осколки сквозь зеленоватое покрывало сверкали огни городов людей и лейвианцев.

Сидя за астронавигационной панелью, затаив дыхание, смотрел на этот пышный мир Алекс Райдер. Ему так и не разрешили сойти на поверхность планеты и этот вид вырвал из его груди вполне отчетливый вздох сожаления. Джейсон Райдер, его отец, сердито хмыкнул и заученным движением коснулся кнопок пульта управления. Кому-кому, а ему хорошо было известно то грустное чувство, которое испытывает астронавт при виде подобного великолепия с орбиты и не имея возможности прикоснуться к его роскоши. Он был однажды на поверхности Лейва. Незабываемое впечатление... Но правила и требования Галактической Кооперации Миров строги и разумны. Лейв, как и любая другая планета, не место отдыха и не объект для любопытства. Это живой, развивающийся мир и в нем живут люди, для которых он является тем же, чем когда-то была Земля для человечества: кровом, матерью, родным домом.

"В другой раз, в другое время" - решил Алекс. Он заслужит себе право посещений Лейва, а сейчас его карьера только начинается. Ему еще так долго учиться.

Райдеры были космическими торговцами в течение трех поколений. Основоположником дела был Бен Райдер. который торговал почти исключительно тем, что снимал с разбитых пиратских кораблей. Бен жил на грани и пришел день звездного года и настала ночь, когда он не вернулся. Ничто не нарушит уединения его могилы, сколь неизвестной, столь и далекой в межзвездной пустоте.

Его сын, а впоследствии и Джейсон Райдер, его внук, продолжили семейное дело. Скоро и Алекс примет главное решение - ставить на карту свою жизнь, снуя челноком между мирами или освоить другую профессию.

Космическая торговля! Давайте честно посмотрим правде в глаза. Это не развлечение для юнцов, одержимых идеей быстрого обогащения. Вы можете всю жизнь возить пищу, оборудование и текстиль и едва-едва наскрести крохи, чтобы купить клочок земли на побережье какой-нибудь планеты земного типа, чтобы провести остаток дней в тишине и уединенном комфорте.

Вот так.

Целая жизнь, пропитанная потом и кровью за дом и чистую голубизну чужого моря у его порога. Конечно, есть и другие пути, если Вы хотите большего: наркотики, рабы, экзотические животные, оружие, повстанцы, - займешься ими и богатство придет наверняка, а вместе с ним и пираты и рейдеры и каперы.

И полиция!!!

Усталость многих лет честной торговли уже сказалась на облике Джейсона, но он всегда по-немногу откладывал, и эта грузовая яхта была предметом ого радости и гордости. Он в любой момент мог прервать свою торговлю и ненадолго отдаться долгожданному отдыху, но справедливо полагал, что пустые трюмы бывают только у тех, у кого и в голове пусто, так что никогда не летал без груза, ради удовольствия. Вот и сегодня он был загружен соком экзотических ягод.

В этот рейс он взял с собой сына. Пусть парень посмотрит, что такое космос, может быть и у него появится интерес к семейному делу, пусть узнает, какова она жизнь в вечном вакууме.

Алекс Райдер был высоким светловолосым юношей. Он был отлично сложен и на своей планете, Онтиате, уже успел стать чемпионом по атмосерфингу. Как и любому другому парню, ему не терпелось поскорее перейти грань, отделяющую ученика от профессионала и начать строить стабильную жизнь: надежная девушка, надежная работа и первые планы по покупке земли.

Впрочем, у него еще был целый год для принятия решения, год серфинга, бейсбола в свободном падении, заоблачных пикников и прочих развлечений. Ему некуда спешить.

Но он еще любил космос, он любил этот солнечный блеск на обшивке корабля, этот грохот космопортов и неизведанность новых миров. Он обожал это чувство исследователя, первооткрывателя.

Голос из системы связи заставил его очнуться: "Авалония! До точки джамп-перехода четыре минуты малым ходом".

"Принято", - ответил Алекс и поднастроил автопилот, отец откинулся в кресле и ласково улыбнулся, ему пока нечего делать.

Опять голос системы управления: "Вход в джамп-переход по каналу два-семь, восток. Сорок пять".

"Принято", подтвердил Алекс и отец начал разворачивать корабль, готовясь к опасному гиперпереходу. " Всё выглядело спокойно.

На кормовом мониторе темная тень наползла на сверкающий диск планеты, - еще один корабль готовится к гиперпрыжку. Алекс не обратил на него особого внимания, сосредоточившись на предстоящем переходе. Отец придирчиво рассмотрел чужой корабль, но расслабился. Все в порядке. Как мог знать он, что жить ему оставалось всего четырнадцать минут.

Выполнить гиперпереход в такой сложной и перегруженной системе, как орбитальный

космопорт Лейва, дело очень непростое. Сотни глаз следят за каждым вашим движением, фиксируя малейшие просчеты. Одна ошибка в орбитальном маневрировании и в другой раз, на совсем другой планете, при подходе к станции "Кориолис" в космическом вакууме перед вами ярко вспыхнет транспарант "ПРИЕМА НЕТ".

Корабль дрейфует, строго подчиняясь указаниям службы мониторинга станции вместе с десятками других кораблей. Разворот - разгон – торможение - вращение, и все это с точностью до секунды: как по углу, так и по времени. Только так можно избежать столкновения хотя бы вон с тем двухтысячетонным торговцем, готовым протаранить двигательный отсек.

Далее управление берет на себя представительство службы Безопасности Полетов. Они проведут ваш корабль в толпе других торговцев, яхт, паромов, челноков, межзвездных лайнеров и стремительных полицейских патрулей. Серебряные стрелы прорезают темноту, ярко вспыхивают зеленые и голубые бортовые огни. Здесь и там мигают предупреждающие огни маяков и на мгновение экраны заслоняются стеной серого металла.

Вы продираетесь сквозь этот хаос и новый голос требует внимания - это служба Управления Навигацией Дальних Переходов. Она выведет вас в точку гиперпрыжка.

В течение нескольких минут вам предстоит преодолеть, скажем, семь световых лет и вы можете подумать, что это огромное пространство, но это не совсем так. Переход происходит по гипертуннелю, а это такой же туннель, как и любой другой. Он отличается лишь тем, что внутри него не существует обычного пространства - это магическое место, в котором обычные законы Вселенной не работают, туннель наводится под постоянным наблюдением и управлением. Здесь и там, через несколько тысяч парсеков, размещены спутники системы мониторинга, работают спасательные станции. Туннели сходятся и разветвляются и все их пространство пронизано сотней каналов, по которым идут корабли. Все сделано для защиты от двух главных опасностей гиперперехода: атомной реорганизации и смещения во времени.

Попробуйте своим ходом совершить гиперпереход хотя бы на пол световых года и вам очень повезет, если вы окажетесь в своей Вселенной.

Вы можете выйти из магического пространства вывернутый наизнанку (не очень приятное зрелище). Деформация пространства может не повредить вашему кораблю, но то желе, которое плавает в кабине - это вы. А еще ходят легенды о том, что полет может пройти и вполне благополучно и, спустившись с орбиты на Землю, вы будете долго недоумевать, чем вы помешали той гигантской ящерице и почему она так сильно переживает от вашего появления в ее славной доисторической пустыне.

Итак, в тот судьбоносный день Алекс Райдер внимательно прислушивался к механическим голосам роботов Службы Управления Навигацией, выводящим его в точку перехода к планете Листи. Он расслабился в кресле рядом с отцом и с интересом следил за работой космопорта, а за кормой нависла тень другого корабля, следующего к туннелю. То был грузовой корабль класса "Кобра".

Никто не знает, как так получилось, что космическим кораблям стали присваивать змеиные имена. Корабль Райдеров был довольно беззащитным "Офидионом", с двумя гипердрайвами и минимальным вооружением, пригодным для уничтожения разве что астероидов, метеоритов, и "сбесившихся кораблей". Так называли корабли неуправляемые или управляемые лихими юнцами ради потехи.

"Кобра" была куда более могучим кораблем.

"Кобры" - обычные торговые корабли, но в большинстве своем они погребены под горой оружия и всевозможнейших защит, которыми оснащают их крепко сбитые суровые капитаны. И на это есть причины…

Быть торговцем - это значит быть опасным и всегда рисковать. Опасным - потому, что если хочешь выжить, то должен хорошо знать свое оружие и как им пользоваться в космическом бою. Ты должен уметь мгновенно распознать пирата, анархиста или таргонский крейсер. Ты должен уметь обойти полицейские ловушки, если на борту есть хотя бы один из тысяч запрещенных видов товара. Рисковать - потому, что нет ничего слаще для корсара, чем жирная "Кобра", набитая мехами, минералами, рудами и текстилем. Быть

торговцем - это значит стрелять первым, а потом молиться, что ты не ошибся и твоя жертва действительно была пиратом. Ошибешься, и никакая броня и никакие ракеты не спасут тебя от "Вайперов".

"Вайперы"! Полицейские корабли. Маленькие, быстрые, смертоносные и невероятно цепкие. Пилот, конечно, человек, но убей человека и корабль продолжит атаку. Уничтожь корабль и будешь иметь дело с ракетой. Уничтожь ракету и всю оставшуюся жизнь шарахайся от каждой тени.

"Вайперы" не кусают. Они впиваются.

... Одиннадцать минут...

- Посмотри, не часто увидишь такое...

Слова отца прервали сосредоточенное изучение планеты, которым занимался Алекс. Справа параллельным курсом к гипертуннелю шел корабль странной формы, мигая мощными бортовыми огнями. Он сверкнул на солнце и Алекс увидел медленно вращающееся рыбье тело.

"Моури". Подводный корабль, способный летать в космосе. Его действительно редко можно было встретить в космосе. На таких планетах, как Регити и Аона, где только верхушки вулканов возвышаются над водой, "Моури" был и грузовым и пассажирским кораблем. Он был важнейшим средством связи с подводными городами.

Алекс с интересом рассматривал необычный корабль, а затем вновь обратился к кормовому экрану.

- Просигналить ей, чтобы держалась подальше?

Джейсон покачал головой, только сейчас Алекс понял, что и отец уже давно следит за этим кораблем. На мостике "Авалонии" возникло напряженное ожидание, это было непривычно и это было неприятно.

Что-то было не так. Алекс не знал, что именно, но все сильнее чувствовал это.

Что-то шло не по раз и навсегда установленному распорядку.

Вспыхнул сигнал, разрешающий вход в гипертуннель, раздался сопровождающий звуковой сигнал, в этот момент жизнь "Авалонии" съежилась до девяти минут.

Вблизи входа в гипертуннель всегда роятся стаи транзитных кораблей. Большинство из них швартуются группами к орбитальным буям. Механики и ремонтники используют эти часы вынужденного простоя для того, чтобы еще раз проверить и подремонтировать внешнее оборудование, в таком месте, на такой перегруженной системе, как Лейв, можно увидеть корабли любого когда-либо выпускавшегося типа.

По мере приближения к туннелю Алекс практиковался в распознавании кораблей - весьма необходимый навык для космического торговца. Распознать не пилотируемые орбитальные челноки было довольно легко, он увидел два "Эспа", принадлежащих военному флоту - небольшие высокоманевренные, смертоносные корабли, прекрасно защищенные от ударов, оснащенные самыми современными боевыми системами. Еще он увидел "Крейт", так называемый "старстрайкер" - маленький одноместный корабль, очень любимый первопроходцами и торговцами.

Справа стыковалась для высадки пассажиров цилиндрическая масса "Анаконды" - массивный грузовик, переделанный под пассажирские перевозки. Это был безобразно некрасивый корабль с распахнутыми приемниками космической пыли на носу.

Можно было составлять каталог. Вот "Боа" крейсерского класса, это "Питоны", а вот мечта охотников за призами "Фер-де-Ленс" - плотно упакованный оружием роскошный дворец.

Большие и малые, "Уормы", "Сайдуиндеры", "Мамбы" ... - все это сверкало, мигало, отражало солнечные лучи своими серо-голубыми телами.

Как всегда, здесь были и рекламные корабли-роботы, предлагающие все, что угодно от "настоящего земного эля с медом фирмы Роганз" до "последнего обеда перед входом в гипертуннель".

- Пошли... Пристегнись...

Джейсон всегда говорил так. Алекс напрягся, хотя на самом деле вход в магическое пространство происходит с ничтожной перегрузкой. Мгновенное чувство головокружения, и

вот перед вами неописуемая картина звезд, разлетающихся многоцветными концентрическими кольцами. Впечатление, как будто звездолет летит сквозь вращающуюся трубу, мгновение, и все кончено. Корабль дрейфует в магическом пространстве, где нет ни места, ни времени. Он пересекает огромные пространства между мирами, а сам в эти секунды находится в мире, который нельзя ни представить, ни описать.

Говорят, что в магическом пространстве есть привидения. Может быть поэтому его и зовут магическим. Время сворачивается, а атомы выворачиваются, гравиволны громоздятся друг на друга. Что-то там движется, что это? Живые формы или тени? Атомы или галактики? - Кто знает, нельзя остановиться и выйти наружу посмотреть, здесь могут работать только дистанционные роботы, управляющие посты, распределительные станции и спасательная автоматика, то, что живет в магическом пространстве навеки останется тайной для людей.

Но привидения там точно есть. По крайней мере, призраки первых кораблей, что вошли в гиперпереход и никогда не вернулись.

Да, привидения… и еще тени… Змеиные тени… "Кобра" нависла над ними…

- Боже, что это…?

Джейсон Райдер стал белее снега.

В этом гипертуннеле он ничего не мог сделать, уклоняясь от другого корабля. Алекс воскликнул.

- Ведь он не знает правил! Может, это новичок?

- Возможно, ответил отец, не отрывая глаз от экранов радаров.

С покрытым испариной лицом Алекс следил за надвигающейся тенью "Кобры".

"Прекрасное оснащение… топливоприемники, контейнеры ракет, дополнительные грузовые пилоны, плоский купол отсека энергетической бомбы… богатый корабль… и смертоносный… ", - мысли неслись стремительно.

- Они же не собираются на нас напасть.

- Черта-с-два они не собираются.

… Три минуты …

(Продолжение следует)



"ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

СПЕКТРУМ В ШКОЛЕ

Сегодня мы предлагаем Вашему вниманию несложную экзаменующую программу, которая может быть использована для проверки элементарных знаний учащихся по любому предмету. Она пригодится и тем, у кого есть маленькие дети. Опыт показал, что при всей ее простоте она привлекает к себе детское внимание. Программа воспринимается ими, как игра "Викторина". Может так случиться, что именно с этой программы Ваш ребенок начнет свой путь в большую компьютеризацию.

Программа управляется от несложной системы меню и не требует никаких инструкций по работе - она самообеспечена. В ней есть два основных режима работы:

1. Режим тестирования.
2. Режим заполнения вопросов и ответов.

После первого запуска командой RUN программа еще не содержит ни вопросов, ни ответов. Сначала вы должны их ввести. Вы можете иметь 5 серий (вариантов) вопросов и ответов на разные темы. Каждая серия содержит 8 вопросов и ответов.

Выбрав, какой вариант Вы хотите заполнить, вводите вопросы и ответы по указанию от компьютера. Закончив заполнение варианта, проверьте его на отсутствие ошибок. Если не все в порядке, Вам надо будет снова войти в этот режим. На этот раз компьютер предложит Вам выбор:

1 - внести изменения

2 - заменить вариант

В режиме тестирования компьютер предложит учащемуся выбрать вариант, с которым он хочет поработать. Разумеется, для выбора будут предложены только те варианты, которые уже заполнены вопросами и ответами.

Учащийся сам может выбрать, в каком порядке он будет отвечать на предложенные вопросы. В случае правильного ответа на экране появляется веселая физиономия и играет веселая музыка. При неправильном ответе и музыка и физиономия становятся печальными.

При вводе программы у Вас будут, конечно, проблемы с вводом символов русского алфавита. Мы не можем сделать это за Вас, поскольку не можем заранее знать Ваших возможностей. Многие работают на компьютерах с русифицированным ПЗУ и имеют команды для переключения с одного шрифта на другой. У некоторых даже есть как бы дополнительный символьный регистр. Если у Вас ничего в этом смысле в компьютере нет, то Вам надо русифицировать компьютер программно. Как это делается, мы уже многократно писали, но должны предостеречь Вас от русификации путем использования символов графики пользователя UDG. Дело в том, что программа уже использует графические символа от "А" до "F" для изображения "физиономий". Эти символы в распечатке программы подчеркнуты и набирать их следует в графическом режиме. Блок данных, задающий "конструкцию" этих символов, расположен в строках 1000. . . 1050.

Может быть, Вы решите отказаться от изображения этих "физиономий" и тогда вернете возможность использования символов UDG для изображения прописных букв русского алфавита. В этом варианте тоже есть своя прелесть, т.к. тогда возможны двуязычные экраны и Вы сможете применить программу для проверки знаний по иностранному языку.

Может быть, Вы захотите организовать несколько банков символов графики пользователя и оперативно переключаться между ними, когда это надо. Как это делается, мы писали в недавно вышедшем первом томе, посвященном графике "Спектрума" - "Элементарная графика" (М:,"ИНФОРКОМ", 1992г., 208 стр.).

Но по всей видимости, Вам лучше русифицировать компьютер полной сменой символьного набора. Из наших последних публикации на эту тему посмотрите пожалуйста статью Алексеева А.Г., посвящённую полной русификации программы "MASTERFILE-09" в "ZX-РЕВЮ-92" на стр. 29-32,71-75. Полезными будут советы этого автора, приведенные и в данном выпуске "ZX-РЕВЮ" в статье "Профессиональный подход".

При вводе и отладке программы обратите особое внимание на следующие обстоятельства:

1. Подчеркнутые символы A...F являются символами графики пользователя UDG и вводиться должны в графическом режиме (курсор G).

2. Только первый запуск программы можно выполнять командой RUN. После того, как вы заполните хотя бы одну серию вопросов и ответов, эту команду уже подавать нельзя, так как по команде RUN обновляется содержимое программных переменных и массивов и Вам придется снова заполнять вопросы и ответы.

При втором и последующем запусках вместо RUN давайте команду GO TO 70.

ВИКТОРИНА

```
10 REM здесь Вы можете разместить
20 REM необходимые Вам
30 REM процедуры, например для
40 REM русификации компьютера.
49 REM
50 DIM h$(5,32): DIM q$(5,8.28): DIM a$(5,8,13): DIM p$(8): DIM t$(5): DIM e$*(1)
60 LET t$="00000"
70 BORDER 7: PAPER 7: INK 0: CLS
80 RESTORE 1000: GO SUB 1000: LET new=0
90 REM *** Начало работы
100 DIM c$(8): CLS
120 FOR y=1 TO 13 STEP 6
130 FOR x=1 TO 29 3TKP 28
140 PRINT AT y,x; INK 4;"AB";AT y+1,x;"CD";AT y+3,x; INK 2; "AB"; AT y+4,x;"EF"
150 NEXT x
160 NEXT y
170 PRINT INK 4;AT 19,1: "AB"; AT 20,1 ;"CD"; AT 19,29; "AB"; AT 20,29;"CD"
180 PRINT AT 8,5; " Выберите режим работы:"
190 PRINT AT 11,4; "1 = Ответы на вопросы"
200 PRINT AT 13,4; "2 - Ввод новых вопросов"
210 PRINT AT 15,4; "3 = Конец работы"
220 PRINT AT 19,5; BRIGHT 1;"Нажмите нужную клавшу"
230 IF INKEY$="1" THEN GO TO 280
240 IF INKEY$="2" THEN GO TO 600
250 IF INKEY$="3" THEN GO TO 900
260 GO TO 220
270 REM
230 REM ** Ответы на вопросы**
281 REM
290 CLS : INPUT "": PRINT AT 0,7; BRIGHT 1; "Подумай и ответь"
300 GO SUB 1300
310 PAUSE 30: PRINT #0;AT 0,0;"Введи номер варианта "
320 GO SUB 1210: LET n=i: IF n>5 THEN GO TO 320
330 IF t$(n)="0" THEN PRINT AT 6,6; FLASH 1; "Этот вариант не готов": INPUT "": PAUSE 150:
     GO TO 90
340 CLS: LET p$=" 12345678"
350 FOR x=1 TO 8
360 LET r=INT (RND*8)+1: IF p$(r)<>" " THEN GO TO 380
370 GO TO 360
380 IF x=r THEN GO TO 360
```

```
390 LET c$(x)=p$(r): LET p$(r)=" ": NEXT x
400 GO SUB 1100: PRINT INK 1;AT 0, 0;h$(n): FOR x=1 TO 8
410 PRINT AT 2*x+2,0;x;AT 2*x+2,2;q$(n,x)(1 TO 14);AT 2*x+3,2;q$(n, x) (15 TO 28);AT
    2*x+2,17; a$(n, VAL c$(x));AT 2*x+2,31;x
420 NEXT X
430 LET p$="12345678": LET t=0
440 INPUT "": PRINT #0;AT 0, 0; "Номер вопроса ?"
450 GO SUB 1210: LET q=i
460 IF p$(q)=" " THEN PRINT #0;AT 0,0; BRIGHT 1;"На этот вопрос Вы уже ответили": BEEP 1,-
    12: PAUSE 50: GO TO 440
470 PRINT BRIGHT 1: AT 2*q+2,2; q$(n,q)(1 TO 14);AT 2*q+3,2;q$(n,q)(15 TO 28)
480 INPUT "": PAUSE 30: PRINT #0; AT 0,0; "Правильный ответ ?"
490 GO SUB 1210: LET a=i
500 PRINT AT 2*a+2,31; FLASH 1;a: PAUSE 50
510 IF VAL c$(a)=q THEN PRINT #0; AT 0.0; "Правильный ответ    " ; INK 4;"AB
    CD": GO TO 530
520 PRINT #0;AT 0,0; "Нет, ответ неверный "; INK 2; "AB                              EF":
    GO SUB 2200: PRINT AT 2*a+2,31;a: GO TO 480
530 PRINT AT 2*a+2,31;a;AT 2*a+2,17; BRIGHT 1;a$(n,VAL c$(a)):GO SUB 2000
540 PRINT AT 2*q+2,2;q$(n,q)(1 TO 14); AT 2*q+3,2;q$(n,q)(15 TO 28); AT 2*a+2,17; a$(n,VAL
    c$(a))
550 LET p$(q)=" ": LET t = t+1: IF t = 8 THEN GO TO 570
560 GO TO 440
570 INPUT "": PRINT#0;AT 0,0; BRIGHT 1; "Нажми любую клавишу"
580 PAUSE 0
590 GO TO 90
599 REM
600 REM**Ввод новых вопросов**
605 REM
610 CLS : PRINT AT 0,7; BRIGHT 1; "Ввод новых вопросов"
620 PRINT AT 2,0; "Эта программа может содержать   до 5 вариантов вопросов и отве- тов.
    Каждый вариант имеет номер"
625 PRINT AT 5,0; "от 1 до 5. Здесь вы можете       узнать, какой вариант готов или внести
    нужные изменения и дополнения."
630 GO SUB 1300: PRINT AT 19,3; "для выхода нажмите клавишу 6"
640 INPUT "": PRINT #0; AT 0,0;" Какой вариант будем заполнять?"
650 GO SUB 1210: LET n=i: IF n>6 THEN GO TO 650
660 INPUT "": IF n=6 THEN GO TO 90
670 IF t$(n)="0" THEN GO TO 790
680 INPUT "Исправление или замена? Нажмите i или z (прочие клавиши - конец."; LINE e$
690 IF e$ = "z" THEN GO TO 790
700 IF e$ <>"i" THEN GO TO 90
710 CLS : GO SUB 1100: PRINT INK 1;AT 0,0;h$(n)
720 FOR x=1 TO 8: GO SUB 880: GO SUB 890: NEXT x
730 INPUT "Какой вопрос будем исправлять? (Клавиша 9 - конец работы)"; x
740 LET x=INT x: IF x<1 OR x>9 THEN GO TO 730
750 IF x=9 THEN GO TO 90
760 PRINT AT 2*x+2,0; BRIGHT 1;x: GO SUB 860
770 PRINT AT 2*x+2,31; BRIGHT 1;x: GO SUB 870
780 LET new=1: GO TO 730
790 CLS : INPUT "Заголовок (до 32 букв)? ", LINE h$(n)
800 GO SUB 1100: PRINT INK 1;AT 0,0;h$(n)
810 FOR x=1 TO 8: GO SUB 860: GO SUB 870: NEXT x
820 LET t$(n)="1": LET new=1
830 INPUT "": PRINT #0;AT 0,0; BRIGHT 1: "Нажмите любую клавишу"
840 PAUSE 0
850 GO TO 90
860 INPUT "Вопрос (до 28 букв)? ", LINE q$(n,x): GO SUB 880: RETURN
870 INPUT "Ответ (до 13 символов)? ", LINE a$(n,x): GO SUB 890: RETURN
880 PRINT AT 2*x+2,0;x;AT 2*x+2,2;q$(n,x)(1 TO 14); AT 2*x+3,2;q$(n,x)(15 TO 28): RETURN
890 PRINT AT 2*x+2,17: a$(n,x);AT 2*x+2,31; x: RETURN
899 REM
900 REM ** Конец программы **
905 REM
910 IF new THEN SAVE "Victorina" LINE 70: PAUSE 30
```

```
920 CLS : STOP
990 REM
995 REM *Данные UDG-графики*
999 REM
1000 DATA "a",7,31,48,96,76,204,192,193
1010 DATA "b", 224,248,12,6,50,51,3,131
1020 DATA "c", 193,192,216,79,99,46,31,7
1030 DATA "d",131,3,27,242,198,12,248,224
1040 DATA "e",193,192,195,71,108,,46,31,7
1050 DATA "f",131,3,195,226,54,12,248,224
1060 FOR x=1 TO 6: READ e$
1070 FOR y=0 TO 7
1080 READ i: POKE USR e$+y,i
1090 NEXT y: NEXT x: RETURN
1095 REM
1100 REM **Дизайн экрана$**
1101 REM
1102 PLOT 0,164: DRAW 255,0
1104 PLOT 0,163: DRAW 255,0
1110 PLOT 0,148: DRAW 255,0
1120 PLOT 0,147: DRAW 255,0
1130 PLOT 0,10: DRAW 255,0
1140 PLOT 0,11: DRAW 255,0
1145 PLOT 0,148: DRAW 0,15
1150 PLOT 131,11: DRAW 0,152
1160 PLOT 132,11: DRAW 0,152
1165 PLOT 255,146: DRAW 0,15
1170 PLOT 11,11: DRAW 0,136
1180 PLOT 244,11: DRAW 0,136
1190 PRINT INK 1;AT 2,4;"ВОПРОСЫ"; AT 2,20; "ОТВЕТЫ"
1200 RETURN 1204 REM
1206 REM *Прием нажатой клавиши*
1208 REM
1210 LET e$=INKEY$
1220 LET i=CODE e$-48
1230 IF i>0 AND i<9 THEN RETURN
1240 GO TO 1210
1299 REM
1300 REM **Готовность данных**
1305 REM
1310 PRINT AT 11,8; BRIGHT 1; " ВАРИАНТ       ГОТОВНОСТЬ "
1320 FOR i=1 TO 5
1330 PRINT AT i+12, 11;i;AT i+12,19;("ГОТОВ" AND t$(i)="1")+("НЕ ГОТОВ" AND t$(i)="0")
1340 NEXT i
1350 RETURN
1999 REM
2000 REM** Правильный ответ**
2005 REM
2010 LET w=0.07
2020 BEEP 3*w,12: BEEP w,16: BEEP 2*w,14: BEEP w,17: BEEP 3*w,16: BEEP 5*w,12
2030 PAUSE 100
2040 RETURN
2199 REM
2200 REM **неправильный ответ**
2205 REM
2210 LET w=0.1
2220 BEEP 1.5*w,7: BEEP w,4: BEEP 1.5*w,0: BEEP w,4: BEEP .8*w,2: BEEP 2*w,2
2230 RETURN
```

BETA BASIC

Продолжение. (Начало см. на стр. 3,47,91,135)

53. SAVE <строка TO строка;> устройство;> имя

SAVE DATA <УСТРОЙСТВО;>имя

См. также DEFAULT <устройство>

В отличие от стандартного БЕЙСИКа, Бета-Бейсик позволяет выгружать не всю программу, а только ее часть, а также выгружать отдельным блоком программные переменные. Параметр <строка TO строка> указывает, начиная с какой строки производится выгрузка и по какую. Если он не указан, то выгружается вся программа целиком.

В форме SAVE DATA этот оператор служит для выгрузки только программных переменных. Если номер устройства, на которое должна происходить выгрузка, не указан, то выгрузка производится на ленту (если ранее оператором DEFAULT не было задано какое-либо иное устройство в качестве основного). Если же номер устройства задан, то выгрузка производится на соответствующий микродрайв (если командой DEFAULT в качестве устройства ввода/вывода не были назначены локальная сеть или последовательный порт RS232).

Примеры

SAVE 10 TO 200; "fragment" -часть программы, начиная с десятой строки по строку 200 включительно выгружается на ленту под именем "fragment".

SAVE 900 TO;"box" - под именем "box"выгружается часть программы, начиная со СТРОКИ 900 и до конца.

SAVE DATA "vars3" - под именем "vars3" выгружаются все программные переменные.

SAVE 20 TO 70;2;"bit" - под именем "bit" на микродрайв номер 2 выгружается часть программы, начиная со строки 20 по 70-ую.

ВНИМАНИЕ!

1. Если Вы захотите загрузить ранее выгруженную часть программы или блок программных переменных, то имейте в виду, что после команды LOAD стирается имевшаяся в компьютере Бейсик-программа, включая и нулевую строку. Блок программных переменных тоже трактуется как программа, не имеющая номеров строк.

Во избежание подобных коллизий Вам целесообразно подгружать ранее отгруженные фрагменты с помощью команды MERGE.

2. Одно из важнейших назначений команды SAVE строка TO строка состоит в том, чтобы Вы могли отгружать свои процедуры по-отдельности и формировать из них на кассете библиотеки процедур для последующего использования. Узнать начальные и конечные номера строк для каждой процедуры можно с помощью ранее рассмотренной команды LIST PROC.

Может быть, вы сочтете целесообразным перед выгрузкой процедуры переместить ее в конец программы, поменяв в ней номера строк с помощью команды RENUM и впоследствии использовать с помощью команды MERGE.

3. Наиболее целесообразное применение оператора SAVE DATA для отгрузки состояния программы. Если вы написали игровую программу, имеющую большую продолжительность, то с помощью такой команды сможете дать пользователю возможность отложить игру. Впоследствии он сможет начать ее сначала или продолжить, загрузив отложенный блок программных переменных.

54. SCROLL код направления <,число> <;x,y; ширина, длина>

Клавиша: S

См. также ROLL

Команда SCROLL имеет синтаксис очень похожий на синтаксис команды ROLL (следует сначала прочитать раздел о команде ROLL). Основное отличие состоит в том, что команда SCROLL может быть использована без параметров, а команда ROLL не может. В этом случае SCROLL вызывает скроллинг экрана на одно знакоместо вверх.

Если за командой стоит код направления 5,6,7 или 8, то текущее окно (а обычно это весь экран) будет передвинуто в заданном направлении (направление смещение стандартно для "Спектрума"):

5 - влево 6 - вниз

7 - вверх 8 - вправо

Когда часть изображения выходит за пределы экрана, она безвозвратно теряется. С противоположной стороны экрана вытягивается чистое поле.

Команда может действовать не на весь экран, а только на заданное окно. В этом случае следует задать параметры.

X, Y - координаты левого верхнего угла окна (задаются в пикселах), система координат та же, что и для команд PLOT и DRAW.

Ширина - размер окна по горизонтали (задается в знакоместах).

Длина - размер окна по вертикали (задается в пикселах).

Обе команды и SCROLL и ROLL широко применяются при разработке игровых программ, а также в различных графических приложениях. Попробуйте поэкспериментировать с теми примерами, которые были приведены для команды ROLL, заменив команду на SCROLL и посмотрите на разницу их действия.

А вот пример небольшой программы, которая передвигает по экрану символьную строку.

```
100 LET a$="HAPPY NEW YEAR"
110 FOR c = 1 TO LEN a$
120 PRINT AT 10,31; INK 7; A$(c)
130 FOR p=1 TO 8
140 SCROLL 5; 0,95;32,8
150 NEXT p
160 NEXT c
170 FOR p=1 TO 255
160 SCROLL 5; 0,95; 32,8
190 NEXT p
```

Обратите внимание на то, что в строке 120 устанавливается белый цвет INK (предполагается, что исходно цвет PAPER тоже белый). В этом случае символы, печатаемые в позиции 10,31 оказываются невидимыми и только по мере смещения влево командой SCROLL проявляются на экране. Печать символов по одному выполняет цикл по "c" (строки 100...160). Смещение их влево на одно знакоместо выполняет первый цикл по "p" (строки 130...160), а второй цикл по "p" (строки 170...190) выводит текст за пределы экрана.

Эти же принципы могут быть использованы для изящной выдачи текстов на экран, например при печати информационных сообщений.

```
200 DATA "Данным давно в далекой"
210 DATA "галактике жили были..."
300 FOR k=1 TO 2: READ a$
310 PRINT AT 21,0; INK 7; a$
320 FOR P=1 TO 8:
330 SCROLL 7
340 NEXT p
350 NEXT k
360 FOR p=1 TO 176
370 SCROLL 7
380 NEXT p
```

55. SORT

или

SORT INVERSE строковый массив
или числовой массив
или символьная строка
Клавиша: M

Команда SORT переорганизует символьные строки или символы или числа в восходящем или в нисходящем порядке. Рассмотрим для начала ее работу с символьными массивами на примере следующей программы, которая генерирует 100 десятибуквенных символьных строк. (Вы можете ускорить работу этой программы, если воспользуетесь вместо стандартной функции БЕЙСИКа RND функцией Бета-Бейсика RNDM, о которой речь пойдет ниже.)

```
100 DIM a$ (100, 10)
110 FOR s=1 TO 100
120 FOR p=1 TO 10
130 LET a$(s,p) = CHR$(RND*25+65)
110 NEXT p
150 NEXT s
160 GO TO 200
170 SORT a$
200 FOR s = 1 TO 100
210 PPINT a$(s)
220 NEXT s
```

Как только массив будет сгенерирован (а это займет определенное время), программа распечатает его в том порядке, в каком он получится.

Теперь дайте прямую команду GO TO 170 (только не RUN, а то массив будет утрачен) и Вы увидите, как тот же массив будет распечатан в алфавитном порядке.

Сортировка 100 строк займет 0.2 секунды и это время очень мало зависит от длины строк, но сильно зависит от их количества. Сортировка массива длиной 200 строк займет примерно 0.7 сек., а для массива в 400 строк - около трех секунд.

Строки сортируются в порядке возрастания кодов первых символов. Если Вы не знаете, какому символу какой код соответствует, то распечатайте себе на память эту таблицу:

```
100 FOR i=32 TO 127
110 PRINT i, CHR$ i
120 NEXT i
```

Если в строке 170 SORT a$ заменить на SORT INVERSE a$, то массив будет отсортирован и распечатан в обратном порядке.

Вы можете сортировать не весь массив, а только его часть, например:

```
SORT a$ (1 TO 20)
```

отсортирует только первые 20 элементов массива, а команда

```
SORT a$(30 TO)
```

отсортирует все элементы, начиная с тридцатого и до конца.

Можно поступить еще хитрее и отсортировать массив не по первому символу, а например по второму и всем последующим.

```
SORT a$ () (2 TO)
```

В этом случае первый символ не будет приниматься во внимание. Как видите, нам пришлось применять скобки дважды.

Команда SORT позволит Вам создавать и эксплуатировать простые, надежные и гибкие базы данных.

Когда мы говорим о базах данных, то массив, о котором шла речь, будем считать файлом, а его символьные строки - записями. В записи можно выделить различные области для разной информации, назовем их полями. Например, первые 20 символов записи отведем для имени вашего партнера. Это будет поле "ИМЯ". Следующие 20 символов отведен для его адреса - поле "АДРЕС", и, наконец еще один символ - для записи возраста - поле "ВОЗРАСТ". Всего на запись уйдет 41 символ.

Встает вопрос, каким образом одним символом выразить двузначный возраст. Это возможно. Одного символа достаточно для выражения возраста от 0 до 255, если сделать так:

```
LET a$(s;41) - CHR$ n,
```

где s - номер записи в Вашей базе, а n- возраст партнера.

Такая форма хранения чисел достаточно проста и экономит память. Но что делать, если нам понадобится хранить более сложную информацию, например размер сберегательного счета. Вы можете воспользоваться следующий приемом:

```
LET a$ (s,41 TO 46) = STR$ b
```

где b - содержимое расчетного счета. Это число будет храниться в виде строки, например: "100" или "22375".

Правда, при этом возникает один недостаток, связанный с тем, что числа будут выровнены по левому полю и сортировка сработает неправильно. Посмотрите, если у Вас есть три записи "9", "75" и "500", то после сортировки они расположатся в порядке:

```
500
75
9
```

Причина в том, что левое поле при сортировке является первич-[1]

Первый выход из положения прост, но неудобен. Вам всегда придется помнить об этой особенности и. вводя числа в свою базу, добавлять необходимое число ведущих нулей:

```
O00009
000075
000500
```

Более грамотный выход - следующий. Прежде, чем заносить данные по полю "СЧЕТ" в массив, программа должна переформатировать их так, чтобы они были выровнены не по левому полю, а так, чтобы их десятичные знаки занимали одинаковые положения. Сотни - под сотнями, десятки - под десятками, тысячи - под тысячами и т.п. Тогда наш пример по результатам сортировки выглядел бы так:

```
  9
 75
500
```

Такое форматирование можно сделать с помощью функции Бета-Бейсика USING$, о которой мы расскажем чуть ниже. Пример ее использования выглядит так:

```
LET a$(s, 41 TO 46) = USING$; ("000.00",b)
```

Теперь вы можете сортировать свои записи по полям "ИМЯ", "АДРЕС", "СЧЕТ". Вы можете, например, отсортировать базу по именам, а затем первые двадцать записей - по размеру счета и т.п. Вы можете сортировать их как в восходящем, так и в нисходящем порядке.

Конечно, вам следует принять меры предосторожности при заполнении базы, чтобы не нарушить ее структуру, т.е. внося имя партнера следует его всегда записывать с первой позиции и никогда не продолжать за двадцатую, иначе нарушится поле "АДРЕС". Впрочем, все меры предосторожности следует делать программно, проверяя вводимые с помощью INPUT данные и тогда ошибки можно исключить.

Мы разобрались с символьными массивами, но команда SORT может работать и со строковыми переменными:

```
INPUT s$: SORT s$: PRINT s$
```

Если Вы здесь по команде INPUT введете строку "Fred Bloggs", то на печать получите " BFdegglors".

Это не выглядит очень полезным, но позволяет работать с числовыми данными в тех случаях, когда они представлены символьными строками, а мы на наших страницах уже неоднократно упоминали о том, что это весьма экономичный способ хранения чисел в памяти компьютера.

Команда SORT может работать и с числовыми массивами, как с одномерными, так и с двумерными. Синтаксис ее применения тот же, что и при работе с символьными массивами. Двумерные массивы мы можем представлять для себя в виде таблиц, в которых первая размерность - это номер ряда, а вторая - номер столбца. Так, команда

```
SORT b (1 TO 20) (2)
```

[1] В оригинале пропущена строка (Прим. OCR)

отсортирует первые двадцать рядов таблицы по второму столбцу. Обратите внимание на то, что мы всегда должны использовать хотя бы одну пару скобок при имени массива b() для того, чтобы компьютер отличал массив от простой переменной b, даже и в тех случаях, когда внутри скобок ничего не стоит.

```
SORT b()
```

Если используется и вторая пара скобок (для указания номера столбца, по которому производится сортировка), в ней должно быть не более одного числа, в отличие от сортировки символьных массивов.

Сортировка числовых массивов идет примерно в четыре раза медленнее, чем сортировка строковых массивов. Это связано с необходимостью иметь дело с интегральной (пятибайтной) формой записи чисел в "Спектруме" (см. "Программирование в машинных кодах"; М:, "ИНФОРКОМ", 1990, 1992).

В отличие от сортировки символьных массивов и строк, при числовой сортировке первыми идут более высокие числа, а затем низкие. Сделать порядок следования чисел возрастающий можно очевидно командой SORT INVERSE.

56. SPLIT (не ключевое слово).

Фактически вместо этой команды вводится символ "<>".

Клавиша SYMBOL SHIFT + W (не в графическом режиме, а в обычном).

Фактически это не оператор языка, а дополнительная возможность редактирования программы. Если Вы редактируете очень длинную строку (она находится в нижней части экрана) и хотите часть ее ввести в программу, а с оставшейся частью продолжить работу, то можете самым первый символом в любом операторе строки поставить символ "<>" и нажать ENTER. в этом случае начало строки до этого символа перейдет в программу со своим номером строки, а оставшаяся часть останется в области редактирования с тем же номером строки и курсором справа от него, чтобы вы могли первым делом изменить его так, как Вам надо.

Например, в нижней части экрана у вас было:

```
10 PRINT "hello": GO TO 10: <> PRINT "goodbye"
```

Если Вы теперь нажмете ENTER, то в листинг программы пойдет:

```
10 PRINT "hello": GO TO 10
```

а в нижней части останется:

```
10 (курсор) PRINT "goodbye"
```

Теперь Вы можете изменить номер строки и, соответственно, отправить эту строку в ту часть программы, в какую хотите, но можете передумать и "подшить" ее к десятой строке с помощью команды JOIN.

57. TRACE номер строки

или

TRACE: оператор: оператор:..

Клавиша: T.

Эта команда относится к разряду отладочных. С ее помощью вы можете запускать БЕЙСИК-программу на выполнение с постоянной распечаткой результатов заданной строки, заданного оператора или заданной переменной. При этом Вы можете изменять (замедлять) скорость исполнения программы.

Существуют две формы оператора TRACE.

1. TRACE номер строки - вызывает переход GO SUB к этой строке перед исполнением любого оператора в Вашей программе. Это не относится к операторам самой этой строки, а то программы бы зациклилась.

2. Вторая форма: TRACE оператор, оператор: ….

В этой случае перед исполнением любого оператора вашей программы исполняется

последовательность операторов, стоящих после TRACE.

И в том и в другом случае, при использовании оператора TRACE Вам доступны две вспомогательные переменные - lino и stat.

LINO - это номер строки, которая сейчас будет выполняться. STAT - номер оператора в этой строке, который сейчас будет исполняться.

При исполнении трассирущей подпрограммы режим TRACE естественно отключается и вновь будет включен по оператору RETURN, который завершает эту подпрограмму. Содержимое подпрограммы можете избрать любое, например:

```
9000 PRINT INVERSE 1: lino:":"; stat: RETURN
```

Введите эту подпрограмму в свою программу и введите команду TRACE 9000 в ту точку программы, с которой хотите начать отладку.

Если Вы хотите воспользоваться второй формой оператора TRACE, то можете не вводить строку 9000, а вставить в ту строку, с которой хотите начать отладку, последовательность операторов:

```
TRACE: PRINT INVERSE 1; lino; ":"; stat: RETURN
```

Отключить режим трассирования, начиная с какой-то строки в Вашей программе можно вставив туда оператор TRACE 0.

Приведенная выше TRACE-подпрограмма позволит получить на экране номера строк и номера операторов, исполняемых Вашей программой в любой момент. Для того, чтобы отличать их от тех данных, которые программа должна выдавать на экран сама по себе, включен режим инверсной печати INVERSE 1.

Если Вы хотите, чтобы не только номера исполняемых строк, но их содержимое было постоянно перед глазами, Вы можете использовать команды:

```
LIST lino TO lino
```

или

```
LIST lino-1 TO lino
```

Если у Вас задан интервал между строк больше, чем 1, вторая команда будет отличаться от первой тем, что при печати содержимого строк не будет изображаться позиция курсора.

Команды RUN и CLEAR также отбивают режим трассирования, как и TRACE 0.

Если Вы хотите замедлить исполнение программы, введите в подпрограмму TRACE оператор PAUSE или сделайте это иным доступным Вам способом, например через BEEP.

Вы можете распечатать также и содержимое интересующих Вас переменных, задав их печать в подпрограмме, но тогда постарайтесь, чтобы эти переменные были объявлены как можно ранее в Вашей программе, иначе можете получить сообщение "Variable not found".

Чтобы отличать ту печать, которую делает оператор TRACE от той, которую ведет сама программа, Вам может быть захочется использовать оператор PRINT AT (а это может пригодиться, если программа строит некоторое графическое изображение и Вы не хотите его нарушать посторонней печатью). Но в этом случае Вам придется позаботиться о том, чтобы перед вызовом трассирующей подпрограммы запоминались координаты текущей позиции печати, а после ее исполнения они бы восстанавливались. Нижеприведенная подпрограмма позволит это сделать:

```
9000 LET POS = DPEEK(23688)
9010 PRINT AT 0,0; lino;":";slat; "     ","a$= "; a$
9020 DPOKE 23688, POS: RETURN
```

Системная переменная SPOSN, расположенная в адресах 23688 и 23689 хранит информацию о текущих координатах позиции печати. Запоминая и восстанавливая ее через переменную POS, Вы добьетесь того, чего хотели.

Поработав какое-то время с Бета-Бейсиком, Вы сами для себя выработаете наиболее удобную отладочную процедуру и тогда сможете присвоить ее с помощью команды DEF KEY какой-либо клавише и отгрузить вместе с самим Бета-Бейсиком на ленту, чтобы впоследствии вызывать одним нажатием тогда, когда надо.

58. UNTIL условие.

Клавиша: K.

Оператор позволяет исполнять циклы DO - LOOP до тех пор, пока "условие" не станет

справедливым.

Подробности см. в описании операторов DO - LOOP.

59. USING, USING$

Клавиша: U.

И USING и USING$ служат для того, чтобы задать формат, в котором Вы хотите распечатать числа. Оператор USING самостоятельно не используется, а употребляется только в качестве квалификатора оператора PRINT или LPRINT, в то время, как USING$ является функцией, имеет самостоятельное значение в виде символьной строки. Функция USING$ выдает символьную строку такой, какой она могла бы быть напечатана при использовании оператора PRINT USING. Благодаря этому появляется возможность использовать форматированные строки не только с командой PRINT, но и с любой другой, например LET, которая может работать со строками.

При использовании USING или USING$ желаемый формат задается в виде форматной строки. Это символьная строка, в которой знаком "хэш" (#) обозначены ведущие пробелы, символ "ноль" обозначает ведущие нули и любой из них может служить для указания значащих цифр после десятичной точки.

```
100 FOR n=1 TO 30:
110 LET X-RND*100
120 PRINT x, USING "###.##";x
130 NEXT n
```

Вы получите на экране два столбца. Слева - неотформатированная печать, а справа - отформатированная. Обратите внимание насколько она выглядит аккуратнее. Попробуйте в строке 120 изменять форматную строку и посмотрите, как будет меняться результат. Обратите также внимание, что при форматировании чисел с помощью USING происходит их округление до заданной в формате десятичной цифры.

А вот еще несколько примеров того, как работает форматная строка для числа 12.3456.

```
"##.#"         12.3
"###.#"        _12.3
"####.##"      __12,35
"000.00"       012.35
"00"           12
"$00.00"       $12,35
"0.00"         %..3
```

Предпоследний пример демонстрирует возможность использования в форматной строке произвольных символов, а не только знаков "#" и "0".

Последний пример демонстрирует случай переполнения формата. Нельзя одним знаком выразить двузначную целую часть - об этом свидетельствует символ %, выводимый на печать, как индикатор ошибки.

Функция USING$ очень похожа на оператор USING. Разницу продемонстрируем на примерах. Вместо

```
PRINT USING a$; number
```

можно сделать

```
PRINT USING$ (a$, number)
```

или

```
LET b$ = USING$ (a$,number) PRINT b$
```

60. VERIFY <строка TO строка;> устройство;> имя

VERIFY DATA <устройство:>имя

Как и стандартная команда VERIFY, эта команда с параметрами служит для проверки правильности выгруженного на ленту или микро-драйв блока данных (программы или ее фрагмента).

Мы не будем останавливаться на синтаксисе команды - он в точности соответствует синтаксису команд SAVE и SAVE DATA, рассмотренных ранее.

61. WHILE условие.

Клавиша: J.

Оператор позволяет исполнять циклы DO - LOOP до тех пор, пока "условие" справедливо.

Подробности см. в описании операторов DO - LOOP.

62. WINDOW номер окна <,x,y,w,l>

Клавиша: 5.

См. также CLS, CSIZE.

Команда WINDOW позволяет Вам назначить часть экрана для выдачи информации при печати или листании. Если окон несколько, то каждое имеет текущую координату позиции печати, а также статусы OVER, BRIGHT, FLSH, INK, PAPER и CSIZE.

В качестве номера окна Вы можете использовать любое число от 1 до 127, причем заданное Вами число не имеет никакого специального значения - это просто опознавательный номер, присущий данному окну.

Все параметры, присвоенные каждому окну, хранятся выше адреса RAMTOP (который, если надо, понижается). Это защищает данную информацию от уничтожения командой NEW и позволяет выгрузить ее на ленту вместе с кодом самого Бета-Бейсика.

Нулевое окно - это весь экран и именно оно является "текущим окном" в момент первого запуска Бета-Бейсика. Это означает, что исходный размер символов и установка цветов - стандартные. Чтобы задать иное окно, Вы можете делать например так:

```
WINDOW 1,0,175,128,176
```

Заданное окно имеет координаты левого верхнего угла 0,175. Ширина окна - 128 пикселов (пол-экрана), а высота - 176 пикселов (весь экран). Текущий установленный размер символов - стандартный (8x8), а установка цветовых атрибутов соответствует нулевому окну, хотя их можно и изменить.

Мы только что задали первое окно, но это еще не сделало его текущим. Чтобы сделать это, надо дать команду WINDOW, указав при ней только номер окна:

```
WINDOW 1
```

Указанное окно стало текущим (если оно было задано). Если же вы забыли его предварительно задать, то получите сообщение об ошибке устройства ввода/вывода - "Invalid I/O device". Единственное окно, которое задается автоматически, без Вашего участия - нулевое окно.

После того, как первое окно стало текущим, весь вывод на печать по командам PRINT, LIST, PLOT, DRAW и т.п. будет производиться только в пределах области данного окна, то есть, только на левой половине экрана. Если хотите поэкспериментировать с окнами другого размера, можете изменить определение первого окна или задать любое другое. В любом случае все изменения, которые Вы произведете, будут видны на экране только после того, как Вы дадите команду WINDOW n, даже если окно с номером n перед этим и так было текущим. Только после этой команды начинают действовать параметры окна, установленные при его задании.

При выходе из окна, а точнее говоря при вызове другого окна, сохраняются все параметры, соответствовавшие данному окну CSIZE, INK, PAPER и т.п., включая и координаты текущей позиции печати. Они сохраняются в областях памяти выше уровня RAMTOP.

Когда вы вновь войдете в это окно, вызвав его командой WINDOW n, все эти параметры будут для него восстановлены. Таким образом, Вы можете менять размер печатаемых символов CSIZE и цветовые атрибуты переключением между окнами.

В нижеприведенном примере задаются два окна и поочередно в них производится печать.

```
10 WINDOW 1,0,175,128,176
20 WINDOW 2,128,151,128,80
30 WINDOW 1: INK 1: PAPER 6
40 WINDOW 2: INK 7: PAPER 1:CSIZE 4,8
50 WINDOW 0
60 PRINT WINDOW 1;"one";WINDOW 2; "two";:GO TO 60
```

Если Вам надо очистить какое-либо окно, Вы можете использовать оператор CLS n, где n - номер окна, подлежащего очистке. В самом начале программы целесообразно давать команду CLS 0, очищая весь экран и подготавливая его к работе.

Если Вам надо удалить из памяти параметры задания какого-либо окна, для этого служит оператор ERASE WINDOW n

63. XOS, XRG, YOS, YRG.

Это не ключевые слова, а своеобразные переменные, с помощью которых можно изменять масштаб экрана и начало координатной сетки, используемой операторами PLOT, DRAW, DRAW TO, CIRCLE, GET и FILL.

XOS - смешение оси X от стандартной.

YOS - смешение оси Y от стандартной.

У этих переменных есть одна особенность. В отличие от прочих, команды CLEAR и RUN их не уничтожают, а устанавливают в заранее заданное фиксированное значение.

Попробуйте дать команду CLEAR, а затеи сделать PRINT XOS или PRINT xos и вы получите "0", а не "Variable not found", как это было бы для обычных переменных.

Оба смещения имеют нулевые значения, если Вы не зададите иные с помощью команды LET.

```
LET XOS=128, YOS=88
```

Такое задание начала координат очень удобно для команды PLOT, т.к. теперь координата X изменяется не от 0 до 255, а от -128 до 127 и, соответственно, координата Y изменяется от -88 до 87.

Команда CLS, как Вы знаете, не только очищает весь экран, но еще и устанавливает текущую позицию печати в координаты 0,0. Если Вы изменили начало координат с помощью xos и YOS, то начальная позиция печати будет устанавливаться после CLS в новое начало координат.

Переменные XRG и YRG определяют масштаб, в котором исполняются команды PLOT, DRAW и пр. Исходное значение XRG = 256 (Вы можете напечатать до 255 различных точек вдоль оси X), а для YRG = 176. Изменив XRG и YRG, Вы меняете масштаб изображения.

```
10 GO SUB 100: REM normal
20 LET XRG = 128: GO SUB 100
30 LET YRG = 88: GO SUB 100
40 LET XRG = 256: GO SUB 100:
50 STOP
100 CLS: PLOT 0,0: DRAW 50,0
110 DRAW 0,50: DRAW -50,0
120 DRAW 0,-50: PAUSE 100
130 RETURN
```

По этой программе сначала рисуется нормальный квадрат, затем он вытягивается по оси X, затем по осям X и Y и, наконец, только по оси Y.

Нижеприведенная программа рисует график функции "синус", используя как смещение начала координат, так и изменение масштаба.

```
100 LET XRG = 2*PI: REM 360 град.
110 LET YRG = 2.2
120 LET YOS = 1.1
130 FOR n=0 TO 2*PI STEP 2*PI/256
140 PLOT n, SIN n: NEXT n
```

Обратите внимание на то, что величина начального смещения начала координат также подвергается масштабированию в заданном диапазоне.

При изображении дуг или окружностей есть одна особенность. Так, последняя точка дуги будет напечатана в соответствии с новой системой координат и с масштабом, но сама кривая - не подвергнется изменениям. То же относится и к окружности. Центр ее будет помещен в точку в соответствии с новой системой координат и с масштабом, но у Вас нет средств промасштабировать радиус, так что если Вы захотите подобным способом получить растянутую вдоль какой-либо оси окружность, то у Вас ничего не получится.

Есть интересная возможность использования масштабирования. Предположим, что вы подготовили программу для того, чтобы получать графическое изображение на полном

экране, после этого Вам захотелось разделить экран пополам (по вертикали) и в правой половине экрана распечатывать листинг программы, а в левой половине - результат ее работы. Это сделать несложно, задав два окна. Но теперь есть проблема в том, что раз графические команды у Вас были рассчитаны на полный экран, то они не смогут работать нормально в его одной половине. В этом случае при переключении на окно Вы можете поменять и масштаб LET XRG = 128.

РАЗДЕЛ 3. ФУНКЦИИ

Бета-Бейсик версии 3.0 имеет более 20 новых функций. Эти функции определены в нулевой строке (которая при листинге не воспроизводится). Там же содержится и указание на машинный код, выполняющий непосредственные расчеты для этих функций.

В программе эти функции существуют как обычные функции, заданные пользователем, а в листинге они являются обычными ключевыми словами. Например, если строка программы содержит FN S$, в листинге это проявляется, как STRING$ и курсор проскакивает это слово за одно нажатие. Функции, заданные пользователем и не являющиеся составной частью Бета-Бейсика, действуют как обычно.

Может случиться так, что Вами ранее уже была подготовлена программа, использующая пользовательские функции, совпадающие с функциями Бета-Бейсика. В этом случае Вам придется поменять обозначения своих функций во избежание конфликта. Это легко можно сделать с помощью команды ALTER.

Ввод имен функций Бета-Бейсика может осуществляться полностью, (если Вы работаете в режимах KEYWORDS 3 или 4) или вводом FN, затем буквы, а затем символа "$" или символа "(". "FN" можно получить нажатием клавиши "Y" в графическом режиме или по буквам. Дополнительные функции Бета-Бейсика не будут работать, если в памяти отсутствует машиннокодовая часть программы Бета-Бейсик. С другой стороны, если будет отсутствовать нулевая строка, то будут потеряны определения функций и Вы получите сообщение об ошибке "FN without DEF". В этом случае остальная часть языка будет работать, но воспользоваться новыми функциями Вы не сможете.

При выгрузке написанной Вами программы нулевая строка выгружается вместе с ней и поэтому если Вы загрузите программу, написанную под Бета-Бейсиком, то нулевая строка будет присутствовать на месте, в то же время, если Вы загрузите программу, написанную ранее в стандартном Бейсике, то нулевая строка погибнет. Поэтому если Вы хотите загрузить в Бета-Бейсик программу, написанную не в нем, то сначала очистите память командой NEW - уберутся все строки, кроме нулевой, а потом подгружайте свою программу с помощью MERGE, что не затронет нулевую строку.

Если же Вы захотите специально удалить нулевую строку, то это можно сделать командой DELETE 0 TO 0.

Ниже мы приводим список новых функций Бета Бейсика.

AND	FN A(
BIN$	FN B$
CHAR$	FN C$
COSE	FN C(
DEC	FN D(
DPEEK	FN P(
EOF	FN K(
FILLED	FN F(
HEX$	FN H$
INARRAY	FN U(
INSTRING	FN I(
ITEM	FN T(
LENGTH	FN L(
MEM	FN M(
MEMORY$	FN M$

MOD	FN V(
NUMBER	FN N(I
OK	FN O(
RNDM	FN R(
SCRN$	FN K$
SHIFT$	FN Z$
SINE	FN S(
STRING$	FN S$
TIME$	FN T$
USING$	FN U$
XOR	FN X(

1. AND (число, число)

FN A (число, число)

По написанию эта функция похожа на обычное ключевое слово AND, но в программе их можно различить по различному синтаксису. Она выполняет побитную операцию AND для двух чисел от 0 до 65535. Если какой-то бит и в первом числе и во втором равен единице, то только в этом случае соответствующий бит результата тоже будет равен единице. Если хотя бы один из них равен нулю, то и в результате этот бит будет равен нулю.

Чтобы лучше понять, как все это происходит, мы воспользуемся функцией BIN$, о которой речь пойдет ниже.

```
BIN$ (254) = "11111110"
BIN$ (120) = "01111000"
BIN$(AND(254,120)) = "01111000"
```

Таким образом, с помощью функции AND можно выключать (маскировать) нежелаемые биты, например:

```
PRINT AND(BIN 00000111, ATTR(line,column))
```

Эта конструкция позволит получить значение INK, установленное для знакоместа с координатами (line, column), а прочие цветовые атрибуты в байте окажутся замаскированы. Поскольку BIN 00000111 = 7, то мы могли записать эту команду и так:

```
PRINT AND (7,ATTR(line,column))
```

Нижеприведенный пример напечатает на экране слово "Bang", если на клавиатуре будет нажата клавиша "F". При этом не имеет значения, какие еще клавиши будут нажаты вместе с ней. В примере клавиатура рассматривается как серия внешних портов. Если Вы не знакомы с тем, как это происходит, читайте нашу книгу ("Программирование в машинных кодах"; М:. "ИНФОРКОМ", 1990, 1992).

```
10 IF AND(BIN 00001000, IN 65022) = 0 THEN PRINT "Bang! ";
20 GO TO 10
```

2. BIN$ (число) FN B$ (число)

Функция дает двоичный эквивалент для заданного числа в виде восьмисимвольной строки, если число меньше 256 или в виде шестнадцатисимвольной строки, если число больше 255, но меньше 65535. Речь идет только о целых числах.

Эта функция очень полезна для тех, кто осваивает программирование в машинных кодах и (или) пользуется функциями Бета-Бейсика для битовых операций AND, OR, XOR.

Кроме того, эта функция может пригодиться при работе с генератором символов, с графикой пользователя UDG и при анализе цветовых атрибутов, системных переменных или при опросе клавиатуры, например:

```
10 PRINT AT 10, 10; BIN$(IN65022)
20 GO TO 10
```

Символьная строка, выдаваемая этой функцией, - это последовательность нулей и единиц. Может быть, вам захочется, чтобы это были иные символы, например, при распечатывании на экране конструкции шрифта знакогенератора очень удобно вместо единиц иметь крестики, а вместо нулей - пробелы, это тоже можно сделать, если принудительно заслать (POKE) в ячейку 62865 код буквы "X", а в ячейку 62869 - код пробела " ".

3. CHAR$ (число)

FN C$ (число)

Эта функция позволяет конвертировать целые числа до 65535 в двухсимвольные строки, что позволяет более экономно использовать память и более компактно хранить данные. Эквивалентом из обычного БЕЙСИКа будет следующая конструкция:

```
LET a = INT(number/256): LET b = number - a*256 LET c$ = CHR$ a + CHR$ b
```

Полученная в результате строка не всегда может быть распечатана, т.к. не всегда в результате получается печатный символ. В этом случае может, например, появиться сообщение об ошибке " invalid colour", если в итоге получился символ, являющийся управляющим кодом. Перед печатью на экран желательно сделать обратную конверсию. Это может сделать функция NUMBER, которая заменит двухсимвольную строку на соответствующее ей число. Поскольку на хранение двухсимвольной строки расходуются только два байта, а не пять, как на число, то мы рекомендуем Вам очень серьезно подумать о том, чтобы использовать эти возможности, если Ваша программа обслуживает и хранит большие массивы чисел (что часто бывает в научных расчетах).

Если Вы не можете работать только с целыми числами, то попробуйте воспользоваться преобразованием их. Например, вам надо хранить число 87.643. Умножив его на 100, вы получите 8764.3. Округлите и преобразуйте 8764 в двухсимвольную строку перед сохранением этого числа в памяти. Когда же оно вновь Вам понадобится, сделайте обратное преобразование, с помощью NUMBER преобразуйте двухсимвольную строку в 8764 и поделите результат на 100. Полученное значение 87.64 имеет неплохую точность по сравнению с исходным числом, эта точность достаточна для большинства практических приложений.

Одним словом, действия над числами Вы можете выполнять как обычно, но перед закладкой числа на хранение Вы его конвертируете, а после вызова вновь восстанавливаете. Это в некоторой степени (в небольшой) снизит скорость расчетов, но очень значительно сэкономит расход памяти. Если рассматривать искусство программирования как вечный компромисс между скоростью операции и расходом памяти, то здесь вам есть поле для деятельности.

Нижеследующий пример демонстрирует последовательность манипуляций по преобразованию числового массива так, как было описано выше.

```
100 DIM a$ (500,2)
110 FOR i = 1 TO 500
120 LET a$(i) = CHR$ (i*10)
130 NEXT i
140 PRINT "Array is ready!"
100 PRINT "Press any key!'
160 PAUSE 0
170 FOR i=1 TO 500
180 PRINT i, NUMBER(a$(i))
190 NEXT i
```

Полученный в результате массив будет занимать только 1 килобайт, в то время, как при хранении чисел стандартным способом его размер был бы 2.5 килобайта.

Оператор SORT работает с конвертированными в символьные строки массивами совершенно правильно.

4. COSE (ЧИСЛО)

FN C (число)

Это функция "косинус". Она отличается от функции COS стандартного БЕЙСИКа тем, что имеет меньшую (но достаточную в большинстве практических приложений) точность, зато выполняется в шесть раз быстрее.

5. DEC (символьная строка)

FN D (символьная строка)

См. также HEX$ (число).

Эта функция преобразует символьную строку, выражающую шестнадцатиричное

число в целое десятиричное число от 0 до 65535. При этом шестнадцатиричное число должно быть выражено символьной строкой длиной от 1 до 4-х символов, а регистр символов - не имеет значения.

DEC ("FF") - 255

DEC ("10") = 16

DEC ("4000") = 16384

DEC ("e") = 14

Эта функция предназначена для обеспечения ввода в программу шестнадцатиричных данных. Например:

```
INPUT a$: POKE address,DEC(a$)
```

Использование "пустой строки" или строки, содержащей символы, не являющиеся шестнадцатиричной формой записи дает сообщение об ошибке "Invalid argument".

(Окончание следует).

Михайленко В. С.

ЗАЩИТА ПРОГРАММ

Продолжение.
(Начало: 9-16, 53-60,97-104, 141-146)

3.2 Работа со встроенными машинными кодами.

В предыдущей статье мы с Вами рассмотрели, как блокировать действие защитных управляющих кодов. (Для этих целей была использована специальная программа, существуют и другие способы просмотра, которым будет посвящена следующая статья данного пособия). Однако просмотра содержимого одного лишь Бейсика бывает явно недостаточно для анализа принципа работы программы в целом и становится необходимым изучение встроенных в Бейсик машиннокодовых процедур. Именно этому вопросу и посвящен данный раздел.

Прежде, чем приступать к работе, нам необходимо определиться с целью, которую вы преследуете при вскрытии процедуры в машинных кодах. Если Вы хотите осуществить какие-либо изменения, то Вам необходимо достаточно кропотливое изучение каждого программного блока. В том случае, если Вы просто изучаете принципы программирования тех или иных авторов, то на первое место по степени приоритетности выступает анализ общей структуры программы.

Здесь мы постараемся рассмотреть оба подхода с тем, чтобы иметь представление о каждом из них. Многие из Вас, уважаемые читатели, наверняка сталкивались с программами, вскрытыми хаккерами. Наиболее известным из них является BILL GILBERT, поскольку программы, вскрытые им, получили наибольшее распространение. И многие из Вас наверняка пытались пробраться сквозь дебри его защиты, чтобы попытаться изменить что-либо, или же просто понять ее принцип действия.

Возможно, что это удалось не всем. Однако, не огорчайтесь. Сегодняшний пример мы построим на исследовании программы, вскрытой этим хаккером, а в последующих выпусках исследуем приемы этого взломщика еще более подробно.

В этой статье мы рассмотрим применение процедур в машинных кодах для защиты программ на примере игры GAME OVER (IMAGINE/DINAMIC). Судя по дате, проставленной хаккером, эта программа была взломана им в 1987 году.

Для того, чтобы с наименьшими усилиями вскрывать программы Билла Гилберта, необходимо уяснить несколько деталей его специфической защиты. В будущем это может очень сильно пригодиться. Так, практически все программы, вскрытые этим хаккером, не имеют защиты от BREAK, т.е. программу можно остановить, нажав клавишу BREAK.

Второе - почти все программы в связи с отсутствием защиты от BREAK, имеют мощную защиту от листинга. В большинстве случаев используются защитные POKES совместно с методом зануления и совместно со встроенными процедурами в машинных кодах. Причем, во многих случаях бывает невозможно работать с программой, не блокировав защитные POKES.

И третье - буквально все программы, вскрытые им, имеют защиту от MERGE, аналогичную описанной Главе 3 первого тома.

Защита от MERGE необходима для того, чтобы не удалось достаточно легко блокировать систему защитных POKES.

Зная эти три особенности, будем грамотно осуществлять взлом, не оставляя без внимания ни одну из них. Наиболее разумным методом ввода первого Бейсик-файла в компьютер является ввод его через программу блокировки автозапуска, рассмотренную в разделе 2.3 второго тома. Тогда защитные POKES ни оказали бы никакого влияния на просмотр листинга данной программы. Однако, можно ввести данный Бейсик-файл и просто подав команду LOAD "" . После его загрузки остановить работу программы нажатием

клавиши break. Теперь весь экран окрасился в черный цвет, а немного ниже середины экрана в прямоугольной рамке можно увидеть надпись:

CRACKED BY BILL GILBERT © 1987

Естественно, что после загрузки данного Бейсик файла с помощью программы блокировки автозапуска надпись в прямоугольной рамке не появится, поскольку ни одна Бейсик-команда не запустилась на выполнение. Для нас это положение выгодное, поэтому в случае загрузки с использованием LOAD "" нам необходимо наверстать упущенное. Когда это будет сделано, Вам будет сообщено дополнительно.

Во первых, сразу после загрузки, желательно подать команду BORDER 7. Это позволит Вам видеть все команды вводимые с клавиатуры. Желательно еще привести в порядок цвета INK и PAPER, подав команды:

```
INK 0: PAPER 7
```

Теперь, подав команду LIST, вы столкнетесь с первичной защитой Билла Гилберта. В данном случае в качестве защиты использована системная переменная 23570. Ее действие немного рассмотрено в Главе 1 первого тома и достаточно основательно в "ZX-РЕВЮ" N10,1991 (стр. 211). Характерным признаком, свидетельствующим об использовании этой системной переменной, является выползание текста снизу вверх вместо раскрутки его сверху вниз, а также появление сообщения:

```
5 OUT OF SCREEN
```

(вне экрана) - во время Ваших попыток автоматического листинга. Как Вам уже вероятно известно, подобный эффект достигается после подачи команды

```
POKE 23570,16
```

Нормальным состоянием данной системной переменной является наличие там числа 6. Поэтому защиту можно блокировать, подав с клавиатуры команду:

```
POKE 23570,6.
```

Только теперь, изменив цвета INK, PAPER и BORDER, а также блокировав действие защитной системной переменной, мы достигли того эффекта, который был получен при использовании для загрузки программы блокировки автостарта. Поскольку все эти изменения осуществила данная программа в ходе своей работы, изменение цветовых атрибутов осуществилось из встроенной подпрограммы в машинных кодах, а POKES, блокирующий защиту, был установлен прямо в Бейсике.

Теперь настало время подробно изучить структуру Бейсик-файла, чтобы через него выйти на программу в машинных кодах - основную цель нашего исследования. Для этих целей используем программу блокировки действия управляющих кодов. Ввиду того, что данный Бейсик-файл имеет значительный объем, необходимо расширить область обработки моей программы. Для этого, после загрузки ее с адреса 62030, необходимо подать команду с клавиатуры:

```
POKE 62032,1
```

Если в будущем Вам и этого окажется недостаточно, можно увеличить это число, либо использовать более точный метод, описанный в предыдущем разделе данной главы.

Программа действует очень быстро и после команды

```
RANDOMIZE USR 62030
```

практически сразу появляется сообщение O.K.

Теперь, после подачи команды LIST, Вы увидите приблизительно такую картину:

```
0>REM FORMAT *BG1987 !REM \NOT
 CONTINUE GO SUB VAL NOT PNOT !NEW
NOT "[\CODE 2\2H\vPIa! RESTORE
STEP ATN !STORE CODE OPEN
#RETURN >STEP ! !PLOT !XCOPY
"[\>x2\> 2\" STEP INT $ STEP INT
$ STEP INT $ COPY STEP INT
$!X'INK <> COPY !!!!AND
))/<>:BCBB:^P^LLIST LLIST > = %CHR$
```

```
""CHR$
   PPFPP^LIST REM
J*FOR BRIGHT RUN DATA POKE >
t c<

   0>REM ! @ GO SUB VAL ICOPY
NOT @! VAL K "SAVE GO SUB VAL NOT
l#

   0>GO TO USR (PEEK VAL
"23635"+VAL "256"*PEEK VAL
"23636"+ VAL "17")
   1 POKE VAL "23570", VAL "16":
LOAD "GAME OVER$"CODE VAL "5E4":
CLS: RANDOMIZE USR VAL "5E4"
    2 LOAD "Game over1 "CODE: LOAD
"Game Over2"CODE: CLS: LOAD "Game
OVER3" CODE: RANDOMIZE USR VAL
"24100"
```

Здесь первые две строки с нулевыми номерами - это встроенные процедуры в машинных кодах. Третья строка, имеющая номер 0, запускает машинный код на выполнение командой:

```
GO TO USR
```

Если вы вызовите эту строку командой EDIT для редактирования и замените GO TO USR оператором PRINT, стерев предварительно номер строки, то получите, что Вы просите компьютер напечатать адрес старта процедуры в кодах. После нажатия клавиши ENTER вы получите число 23772. Это и есть адрес запуска встроенной процедуры в кодах. Именно туда идет реальное обращение Бейсик-программы, т.к. это первая реально выполняющаяся строка, ввиду того, что все предыдущие начинались командой REM.

Для того, продисассемблировать данный машинный код, необходимо воспользоваться специальной системной программой, предназначенной для этих целей. Я использовал программу MONITOR 48, причем открыл в ней дополнительный режим, намного упростивший работу в данном конкретном случае, а в принципе он упрощает процесс изучения и отладки любых программ в машинных кодах.

* * *

ПРИМЕЧАНИЕ

В большинстве справочной литературы к программе MONITOR 48, данный режим работы этой программы не описан. Отсутствует эта информация и в фирменной инструкции, перевод которой был подготовлен "ИНФОРКОМом". Поэтому сегодня в конце раздела приведена специальная статья с описанием этого нового режима работы.

Поскольку сегодня мы работаем с программой, вскрытой Биллом Гилбертом, то надо указать на одну из особенностей его программирования встроенных в Бейсик машиннокодовых процедур. Перед началом работы, вся подпрограмма в машинных кодах переносится в область памяти, начиная с 50000 и лишь потом она начинает работать уже в этой области оперативной памяти.

Не является исключением и наш случай, программа в машинных кодах начинается следующими командами:

```
5CDC          LD HL,5CEA
5CDE          LD DE,C350
5CE2          LD BC, 03E8
5CE5          LDIR
5CE7          JP C350
5CEA          LD HL, C3E6
```

Как видно из приведенной выше распечатки, строки 5CDC - 5CE2 подготавливают регистры процессора для переноса блока памяти командой LDIR. После этого

осуществляется безусловный переход на начальный адрес уже перемещенного блока, т.е. 50000 (C350H). Строка 5CEA дана для того, чтобы Вы имели представление о точке, с которой начинается перенос кодов на новое место

Необходимый для понимания принципа работы программы дисассемблер приведен ниже уже для новых адресов.

```
C350   21E6C3          LD HL,LC3E6
C353   227B5C          LD (#5C7B),HL
C356   AF              XOR A
C357   328D5C          LD (#5C8D),A
C35A   32485C          LD (#5C48),A
C35D   76              HALT
C35E   01A761          LD BC,#61A7
C361   210313          LD HL,#1303
C364   E5              PUSH HL
C365   CDB71E          CALL #1EB7
C368   21761B          LD HL,#1B76
C36B   E5              PUSH HL
C36C   AF              XOR A
C36D   D3FE            OUT (#FE),A
C36F   3E02            LD A, #02
C371   CD0116          CALL #1601
C374   21C9C3          LD HL,C3C9
C377   7E        LC377 LD A,(HL)
C378   FEFF            CP #FF
C37A   CA81C3          JP Z,LC381
C37D   D7              RST #10
C37E   23              INC HL
C37F   18F6            JR LC377
C381   2158FF    LC381 LD HL,#FF58
C381   227BC5          LD (#5C7B),HL
C387   3E78            LD A,#78
C389   3E905C          LD (#5C90),A
C38C   3E20            LD A,#20
C38E   32915C          LD (#5C91),A
C391   0E36            LD C,#36
C393   060E            LD B,#0E
C395   CDE522          CALL #22E5
C398   0E9B            LD C,#9B
C39A   0600            LD B,#00
C39C   1E01            LD E,#01
C39E   1600            LD E,#00
C3A0   CDBA24          CALL #24BA
C3A3   0E00            LD C,#00
C3A5   060B            LD B,#0B
C3A7   1E00            LD E,#00
C3A9   1601            LD D,#01
C3AB   CDBA24          CALL #24BA
C3AE   0E9B            LD C,#98
C3B0   0600            LD B,#00
C3B2   1EFF            LD E,#FF
C3B4   1600            LD D,#00
C3B6   CDBA24          CALL #24BA
C3B9   0E00            LD C,#00
C3BB   060B            LD B,#0B
C3BD   1E00            LD E,#00
C3BF   16FF            LD D,#FF
C3C1   CDBA24          CALL #24BA
C3C4   215827          LD HL,#2758
C3C7   D9              EXX
C3C8   C9              RET
C3C9   161307    LC3C9 DEFB #16,#13,#07
C3CC   110210          DEFB #11,#02,#10
C3CF   071301          DEFB #07,#13,#01
C3D8   909192          DEFB #90,#91,#92
```

```
C3D5  939495          DEFB #93,#94,#95
C3D8  969798          DEFB #96,#97,#96
C3DB  999A9B          DEFB #99,#9A,#9B
C3DF  9C9D9E          DEFB #9C,#9D,#9E
C3E1  9FA0A1          DEFB #9F,#A0,#A1
C3E4  A2FF            DEFB #A2,#FF
C3E6  000000    LC3E6 DEFB #00,#00,#00
C3E9  000000          DEFB #00,#00,#00
C3EC  000000          DEFB #00,#00,#00
C3EF  1D2121          DEFB #1D,"!","!"
C3F2  21211D          DEFB "!","!",#1D
C3F5  0000C6          DEFB #00,#00, #C6
C3F8  29292F          DEFB ")",")","/"
C3FB  C92900          DEFB #C9, ")",#00
C3FE  003A            DEFB #00,":"
```

Если Вы внимательно изучите приведенный дисассемблированный код, то достаточно быстро поймете, что это не вершина программирования в машинных кодах. Хотя следует признать, что подход, примененный в этой программе достаточно оригинален. Я не буду подробно излагать здесь описание данной процедуры, а лишь немного расскажу о ней, чтобы Вы имели обшее представление о принципах ее работы.

После необходимых подготовительных процедур осуществляется печать текстовой информации. Причем любопытен сам метод формирования этой текстовой информации. Данный текст сформирован с помощью UDG графических символов пользователя. Именно в них содержится надпись CRACKED BY BILL GILBERT. Это объясняет и необычно мелкий шрифт данного сообщения. Кроме того, данный прием позволяет обойти попытки будущих взломщиков найти, где находится этот текст с помощью средств поиска заданной символьной строки, которые есть в любой отладочной программе.

Теперь Вы имеете доступ к изменению данного сообщения (разумеется, исключительно из чистого любопытства). Причем, следует отметить, что банки UDG символов не переносятся в верхние области памяти, а всего лишь изменяется значение системной переменной, указывающей на месторасположение этих символов.

После распечатки текста идут процедуры вычерчивания линий. Поскольку это прямоугольная рамка, то линий достаточно 4. Эти процедуры легко обнаружить, как четыре почти однотипных блока.

Как видите, нет ничего сложного. Я надеюсь, что приведенная информация развеет миф о надежности защиты Билла Гилберта. Хочется также надеяться, что наши уважаемые читатели не будут злоупотреблять полученной информацией (об этом мы уже писали в Главе 2 первого тома).

В заключение надо добавить, что здесь рассмотрен лишь один из по меньшей мере трех известных автору способов защиты информации, практикуемых Биллом Гилбертом. Практически аналогичные приемы применены во вскрытых им EXOLON, DEATH WISH 3 и др. А в программах EAGLES NEST и SAS, вскрытых этим хаккером, реализован уже совершенно иной принцип защиты, несмотря на схожесть картинки с сообщением. К более подробному рассмотрению типов защиты программ у Била Гилберта мы еще вернемся.

Глава 4. Изучение блоков в машинных кодах.

4.1 Введение.

Как Вы уже вероятно догадались, данный раздел посвящен вскрытию блоков, целиком записанных в машинном коде. Наши читатели по-видимому понимают, что за изучением блоков машинного кода последует самостоятельное программирование в машинном коде, а это уже прицепиально новый уровень Вашей работы с компьютером. Только поднявшись до этого уровня Вы сможете полностью почувствовать всю мощь и быстродействие Вашей машины, только здесь Вы сможете полностью использовать все ее возможности.

При изучении программирования в машинных кодах, вам обязательно захочется исследовать какие-либо фирменные программные разработки, чтобы узнать новые приемы в программировании и обогатить свой арсенал. Но большинство программ имеют

высококлассную защиту, обойти которую не так просто.

А вот еще одна более реальная ситуация. Предположим, Вы открываете "ZX-РЕВЮ" или другую специальную литературу и видите POKES, т. е. информацию о том, как ввести в игру бессмертие, изменить количеству оружия и пр. Но во многихслучаях первые попытки ввести POKES в программу ничего не дают.

Почему? Да потому, что большинство программ, в которые приятно поиграть на досуге, имеют высококлассные загрузчики в машинных кодах, которые и запускают загруженную программу. А Ваши попытки изменить содержимое ячеек, в которые впоследствии будет загружаться программа, естественно, ни к чему не приведут. И в этом случае нельзя винить авторов, давших POKES. Они изложили всю необходимую информацию, а теперь вашей задачей является правильно ею воспользоваться.

Данная статья имеет среди прочих и цель научить вас, уважаемые читатели, самостоятельно пользоваться необходимой информацией как для изучения новых приемов программирования, так и для установки необходимых POKES. Причем последний вопрос настолько актуален сейчас, что мы рассмотрим два примера предлагаемых POKES непосредственно из "ZX-РЕВЮ".

4.2 Адаптация фирменных программ под индивидуальный вкус.

4.2.1 Адаптация программы GREEN BERET.

Если Вы внимательно читаете "ZX-РЕВЮ", то вам должно быть известно, что такое POKES и для чего они применяются. Однако, если Вы внимательно изучите таблицы POKES, то обратите внимание, что в большинстве случаев предложено лишь значение ячейки памяти и ее содержимое. Собственно говоря, больше Вам ничего знать и не понадобится за одним исключением - необходимо знать как осуществить изменение содержимого указанных ячеек памяти на необходимые значения.

Проблема эта возникает в связи с тем, что почти все хорошие программы имеют загрузчик в машинных кодах и с помощью функции Бейсика POKE Вам ничего достичь не удастся. Не всегда помогает и COPY COPY, поскольку не все программы имеют начальный адрес загрузки 23896, а чтобы узнать этот начальный адрес опять-таки необходимо изучать загрузчик в кодах. Конечно, хорошо бы иметь DISK-MONITOR, позволяющий прервать работу программы и, осуществив необходимые изменения, вновь возвратиться в нее. Однако не у всех читателей он есть и не все собираются в перспективе его устанавливать.

Итак, рассмотрим в качестве примера программу GREEN BERET, одну из лучших "стрелялок" для "ZX SPECTRUM", которая несмотря на некоторую политическую окраску достаточно популярна и у нас в стране. Однако многим читателям явно не хватает предоставленных игрой трех жизней и трех выстрелов из огнемета. Казалось бы, если изменить хотя бы один из этих параметров, то игру удастся одолеть без труда, не говоря уже о том случае, когда удастся изменить их оба сразу. Причем, что самое любопытное - вся необходимая для этой цели информация есть в "ZX-РЕВЮ" (см. N7,8,10 - 1991). Остается лишь изменить содержимое указанных там ячеек, но именно здесь и возникает проблема, поскольку в программе применен собственный загрузчик в машинных кодах, который и осуществляет потом загрузку и запуск программы.

Для того, чтобы правильно вести работу, Вам в первую очередь понадобится узнать структуру всей программы. Это можно осуществить, загрузив ее в один из копировщиков, например ZX COPY 87, TF COPY и т. д. Информация для GREEN BERET, полученная таким образом на копировщике ZX COPY 87:

```
01 GREENBERET  BASIC        10        67
02                                    67
03 T.P.15006   CODE      33475       794
04                                   794
05                                    17
06                                  6912
07                                 10303
08                                 30832
```

Здесь Вы видите, что небольшая Бейсик-программа загружает и запускает загрузчик в машинных кодах, который и осуществляет дальнейшую загрузку файлов, а в конце концов их запуск. Причем, как видим, файлы, загружавшиеся после загрузчика, идут без хэдера, т. е. правильная загрузка их без "родного" загрузчика практически невозможна, поскольку именно он знает адрес памяти, с которого необходимо осуществить загрузку.

Любопытна судьба блока кодов длиной 17 байтов, идущего сразу после загрузчика. Если Вы внимательно изучите программу-загрузчик, то достаточно быстро поймете, что этот коротенький файл в перспективе затирается более мощным программным блоком. Т.е. фактически для работы программы он не нужен. Кроме того, если Вы обратили внимание, он загружается вызовом специальной подпрограммы, загружаемой вместе с загрузчиком. Я думаю, что это не случайно, как не случаен и тот факт, что длина этого блока совпадает с видимой длиной "хэдера". По моим предположениям этот блок служил для защиты от копирования, поскольку некоторые версии копировщиков принимали его за "хэдер", после чего не могли правильно интерпретировать его содержимое и копирование становилось невозможным. Однако появившиеся в последние годы польские копировщики ZX COPY 87, TF COPY, TF COPY-2 свели на нет данный метод защиты, отголоски которого мы можем наблюдать в этой программе, однако вернемся к ее более подробному исследованию, поскольку именно оно поможет нам осуществить желаемые изменения.

Ниже представлен листинг Бейсика данной программы. (Сразу следует оговориться, что версий программы GREEN BERET, распространенных в нашей стране по-видимому несколько. Здесь рассматривается версия, вскрытая TOMI & DALI. Об этом свидетельствует надпись слева на картинке CRACKED BY TOMI & DALI.

GREEN BERET LLIST

```
10 BORDER 0: PAPER 0: INK 0: CLEAR 65535
20 LOAD ""CODE: RANDOMIZE USR 34132
```

Как видим, данная программа на Бейсике достаточно проста и не имеет никакой защиты. Это может объясняться тем, что используется загрузчик в машинных кодах и авторы настолько уверены в психологическом барьере, отпугивающем пользователей от машинных кодов, что не потрудились даже установить какую-либо защиту.

С другой стороны, возможно, что защита БЕЙСИКа все же была, но ее уже снял кто-то из взломщиков.

Наиболее ценной информацией, которую мы почерпнули из данной Бейсик-программы является то, что загрузчик в машинных кодах запускается с адреса 34132.

Рассмотрим теперь дисассемблер загрузчика, начиная со стартового адреса. Необходимо сразу предупредить читателя, что встроенный загрузчик не рассматривается здесь как таковой, а исследуется лишь как подпрограмма, к которой осуществляется обращение. Нам необходимо знать, в какой последовательности загружаются программные файлы и благодаря этой информации найти в программе место, с которого осуществляется запуск программы на выполнение.

Правильная адаптация программы и предполагает именно проведение всей загрузки, а перед самим запуском внесение изменений в машинный код, т.е. исполнение POKES.

На этом пути Вас ждут многие проблемы, но основные среди них - поиск точки запуска программы и правильное внесение изменений, сопряженное с программированием в машинном коде.

Программа-загрузчик использует для своей работы две процедуры - встроенную в ПЗУ и загружаемую в компьютер. Имея перед собой информацию, полученную из копировщика, Вы легко разберетесь, как работает программа.

Фальшхэдер загружается специальной процедурой программы загрузчика. Больше данная процедура нигде не используется. За это отвечают строки 34147 - 34157.

Как видим, данная процедура мало чем отличается от стандартной. Она загружает 17 байтов, начиная с ячейки памяти 36864, о чем свидетельствуют значения, загружаемые в регистры DE и IX соответственно.

Следующий блок программы 34160 - 34174 осуществляет загрузку экрана. Об этом свидетельствует длина файла 6912 и начало загрузки с адреса 16384.

После этого исполнение программы "передается" на адрес 33639 (информация о дисассемблере данной области приведена ниже).

Здесь осуществляется загрузка третьего и четвертого блоков кодов с использованием встроенной процедуры, расположенной в ПЗУ по адресу 1366.

Далее в строках 33665-33670 изменяется содержимое одной из ячеек памяти, после чего управление возвращается на 34180.

После включения прерывания командой IM 1 осуществляется корректировка содержимого памяти, т.е. перенос содержимого ячеек из одной области в другую. Это осуществляется с использованием команды LDIR. по всей видимости, это тоже один из видов защиты.

Далее идут команды изменения содержимого отдельных ячеек памяти и, наконец, переход JP 32768.

Это и есть переход в программу для запуска игры, поскольку все блоки кодов уже загружены, и кроме этого осуществлены необходимые защитные корректировки. Здесь этот безусловный переход осуществляется в "новую" область, т.е. в один из блоков загруженной программы. Однако, не всегда это может оказаться так. Чтобы быть уверенным в достоверности полученной информации, Вам необходимо проверить свое предположение на практике. Проверка показала, что высказанное здесь предположение с успехом подтвердилось.

```
34132 F3                 DI
34133 31FFFF             LD SP,65535
34136 310058             LD HL, 22528
34139 010300             LD BC, 00003
34148 3600         L855E LD (HL),#00
34144 23                 INC HL
34145 10FB               DJNZ L855E
34147 37                 SCF
34148 3EFF               LD A,#FF
34150 DD210090           LD IX,36864
34154 111100             LD DE,00017
34157 CDE684             CALL 34082
34160 DD210040           LD IX,16384
34164 37                 SCF
34165 11001B             LD DE,06912
34168 215884             LD HL,33860
34171 225684             LD (33878),HL
34174 CD5605             CALL 01366
34177 C36783             JP 33639
34180 B7                 OR A
34181 00                 NOP
34182 00                 NOP
34183 00                 NOP
34184 ED47               LD I,A
34186 215827             LD HL,10072
34189 D9                 EXX
34190 FD213A5C           LD IY,23610
34194 ED56               IM 1
34196 2I4083             LD HL,33600
34199 1141S3             LD DE,22601
34202 015402             LD BC,00596
34205 3600               LD (HL),#00
34207 EDB0               LDIR
34209 AF                 XOR A
34210 320A80             LD (32778),A
34213 320B80             LD (32779),A
34216 C30080             JP 32768
34219 D1 POP             DE
34220 7C LD              A,H
34221 FE01               CP #01
```

```
34223 F5                PUSH AF
34224 AF                XOR A
34225 E638              AND #38
34227 0F                RRCA
3422B 0F                RRCA
34229 0F                RRCA
34230 D3FE              OUT (#FE),A
23232 F1                POP AF
34233 D8                RET C
34234 C34083            JP 33600
34237 14                INC D
34238 282A              JR Z,34282
34240 1EFF              LD E, #FF
34242 1A                LD A,(DE)
34243 1000              DJNZ L85C5
34245 5B          L85C3 LD E,E
34246 3F                CCF
34247 2810              R Z,L85D9
34249 00                NOP
34250 88                ADC A,B
34251 1F                RRA
34252 4E                LD C,(HL)
34253 00                NOP
34254 00                NOP
34255 00                NOP
3425E 27                DAA
34257 23                INC HL
34258 00                NOP
34259 00                NOP
34260 00                NOP
34261 00                NOP
34262 00                NOP
34263 00                NOP
34264 D0                NOP
34265 00          L85D9 NOP
34266 00                NOP
33639 DD21005B          LD IX,23296
33643 113F28            LD DE,10303
33646 37                SCF
33647 3EFF              LD A,#FF
33649 CD5605            CALL 01366
33652 DD21DD85          LD IX,34269
33656 11147A            LD DE,31252
33659 37                SCF
33660 3EFF              LD A,#FF
33662 CD5605            CALL 01366
33655 3E00              LD A,#00
33667 323385            LD (34099),A
33670 3A3385            LD A,(34099)
33673 C38485            JP 34180
```

Ниже приведены изменения, которые необходимо сделать, чтобы осуществить одновременное изменение количества жизней и оружия. Рассмотрим эту информацию более подробно.

```
34216 03CB85            JP L85CD
34219 D1                POP DE
34220 7C                LD A,H
34221 FE01              CP #01
34223 F5                PUSH AF
34224 AF                XOR A
34225 E638              AND #38
34237 0F                RRCA
34228 0F                RRCA
34229 0F                RRCA
34230 D3FE              OUT (#FE),A
```

```
34232 F1            POP AF
34233 D6            RET C
34234 C340B3        JP 33600
34237 14            INC D
342338 282A         JR Z,34282
34240 1EFF          LD E,#FF
34242 1A            LD A,(DE)
34243 1000          DJNZ L85C5
34245 5E      L85C5 LD E,E
34246 3F            CCF
34247 2810          JR Z,L85D9
34E99 00            NOP
34250 88            ADC A,B
34251 1F            RRA
34252 4E            LD C,(HL)
34253 21ECB6  L85CD LD HL,46828
34256 77            LD (HL),A
34257 23            INC HL
34256 77            LD (HL),A
34259 23            INC HL
34260 77            LD (HL),A
34261 215CA4        LD HL,42076
34264 77            LD (HL),A
34265 C30080  L85D9 JP 32766
34268 85            ADD A,L
34269 00            NOP
```

Как нам уже известно, строка 34216 осуществляет запуск игры, т.е. к этому моменту загрузка закончилась и закончился также необходимый перенос информации в нужные ячейки командой LDIR. Это именно та точка, которая нужна для осуществления необходимых изменений. Теперь будем рассуждать с точки зрения рядового пользователя компьютером. Если по адресу 34216 осуществляется запуск игры, то значит команды, размещенные в ячейках 34219-34252 просто не нужны. Однако, с другой стороны, весьма маловероятно, что они являются "мусором". Не разбираясь глубоко в принципе работы программы, можно предположить, что они используются загрузившейся программой в процессе ее работы, либо же служат, как переменные загрузчика. Такие логические рассуждения необходимы для того, чтобы безошибочно определить место размещения будущей специальной подпрограммы, изменяющей содержимое ячеек памяти. Как видим, наиболее приемлемым местом является область 34253-34268, поскольку здесь есть немного "пустого" места и мы ножем не опасаться, что чему-либо помешаем.

Верхний предел связан с тем, что с адреса 34269 начинается загрузка блока машинных кодов, - информация, которую Вы могли узнать, внимательно ознакомившись с загрузчиком. Теперь нам необходимо достаточно экономно расходовать память при составлении своей процедуры изменения, чтобы уложиться в отведенное пространство 13 байтов.

Как вам уже вероятно известно, в этой программе необходимо внести следующие изменения:

- для бесконечного оружия

```
46328,0
46329,0
46330,0
```

- для изменения количества жизней

```
42076,0
```

Машинный код, который выполнит необходимые изменения. Вы можете видеть в строках 34253 - 34265.

Таким образом, в том месте, где загрузчик должен запускать игру, мы поставили "жучка", который запускает процедуру для изменения количества жизней и оружия, расположенную по адресу 34253. А после осуществления необходимых изменений в ячейках памяти процедура сама запускает игру на исполнение.

После таких изменений игра становится даже не интересной - Вы можете дойти до

конца буквально за 10 минут. Поэтому мы рекомендуем Вам просто изменить количество выстрелов из огнемета - тогда в игру станет играть чуть легче, но не настолько, чтобы быстро потерять к ней интерес.

Для тех, кто решит сохранить данные изменения, рекомендуем записать информацию на магнитофон, подав команду:

```
SAVE "G.B.CORR." CODE 33475,794
```

Для тех читателей, которые не имеют ни малейшего представления о машинных кодах Z80 и не желают их изучать, Я предлагаю измененный вариант Бейсик-загрузчика, который перед запуском программы в кодах вносит в нее все вышеописанные изменения.

```
10 BORDER 0: PAPER 0: INK 0:CLEAR 65535
20 LOAD ""CODE
30 POKE 34217,205: POKE 34218,133
40 FOR i=34253 TO 34267
50 READ A: POKE I,A
60 NEXT I
70 DATA 33,236,182,119,35,119,33,92,164,119,195,0,128
80 RANDOMIZE USR 34132
```

4.2.2. Новые возможности программы "RENEGADE".

Если Вы внимательно изучили содержимое предыдущего раздела и достаточно подробно разобрались с предложенной вашему вниманию программой, то теперь должны достаточно хорошо разбираться в подобного рода вопросах, поскольку программа GREEN BERET имеет достаточно сложный загрузчик. Однако, возможно, что не у всех хватило терпения и желания подробно изучить предложенное выше описание и потому для тех, кто просто пролистал материал предыдущего раздела, мы предлагаем разобраться с более простой программой RENEGADE. (Конечно, слово "простой" не относится к программе, а только к ее загрузчику).

В этой программе, как и в GREEN BERET не требуется много размышлять, но эта игра достаточно популярна. Здесь так же, как и в GREEN BERET количества жизней, предложенных вначале, бывает недостаточно для прохождения программы до конца.

Бейсик-загрузчик имеет вид:

```
10 POKE 23624,71: POKE 23693,71:
     CLS: LOAD ""CODE: LOAD "SCREEN$:
     RANDOMIZE USR 64000
```

Запуск машиннокодового загрузчика выполняется, как видите, с адреса 64000. Посмотрим, что там содержится.

```
64000 DD21005B          LD IX,23296
64004 11009F            LD DE,40704
64007 37                SCF
64008 3EFF              LD A,#FF
64010 CD5605            CALL 01366
64013 31FFFF            LD SP,65535
61016 C3CB5C            JP 23755
64019 32485C            LD (23624),A
64022 CD6B0D            CALL 03435
64025 21003D            LD HL,15616
61028 113000            LD DE,00048
64031 00                NOP
64032 00                NOP
```

Если Вы внимательно изучите и эту короткую программу, то достаточно быстро поймете, что в адресах 64000-64010 осуществляется непосредственная загрузка главного блока (как Вам уже вероятно известно, в программе RENEGADE загружаются два блока в машинных кодах). Однако нас сейчас интересует только первый, являющийся

машиннокодовым загрузчиком.

Почти сразу после загрузки главного блока осуществляется безусловный переход по команде: JP 23755.

Чтобы осуществить необходимые изменения, нам необходимо воспользоваться данным переходом. Вместо того, чтобы переходить на адрес 23755, мы перейден на адрес 64031, где разместим программу, осуществляющую необходимые изменения, и, осуществив их, перейдем на адрес 23755.

Измененный загрузчик для RENEGADE, набранный в ассемблере "EDITAS-48".

```
10          ORG 64000
20          LD IX,23296
30          LD DE,40704
40          SCF
50          LD A,255
60          CALL 1366
70          LD SP,65535
80          JP NEW
90          LD (23624),A
100         CALL 03435
110         LD HL,15616
120         LD DE,0048
130   NEW   LD HL,41047
140         LD A,162
150         LD (HL),A
160         LD HL,30301
170         LD A,195
180         LD (HL),A
190         JP 23755
200         END
```

Перед вами дисассемблер измененного загрузчика. Эта версия проверена автором и вполне работоспособна.

ПРИМЕЧАНИЕ.

В "ZX-РЕВЮ" N7,8 за 1991г. была дана информация об изменении количества жизней и времени в программе RENEGADE. Однако приведенная там БЕЙСИК-программа не работает с имеющейся у автора версией программы. Для тех, у кого тоже есть такие проблемы дается другой вариант адаптации. Подчеркнуты те байты, которые нуждаются в изменении.

```
64000 DD 21 00 5B   ]!.[
64004 11 00 9F 37   ...7
64008 3E FF CD 56   >.MV
64012 05 31 FF FF   .1..
64016 C3 1F FA 32   C.z2
64020 48 5C CD 6B   H\MK
64024 0D 21 00 3D   .!.!
64028 11 30 00 21   .0.!
64032 57 A0 3E B6   W >6
64036 77 21 5D 76   w!]v
64040 3E C3 77 C3   >CvC
54044 CB 5C 00 00   K\..
64046 00 00 00 00   ....
64052 00 00 00 00   ....
64056 00 00 00 00   ....
64060 00 00 00 00   ....
64064 00 00 00 00   ....
64068 00 00 00 00   ....
64072 00 00 00 00   ....
64076 00 00 00 00   ....
```

Новый же БЕЙСИК-загрузчик, способный внести эти изменения выглядит так:

RENEGADE NEW LOADER LLIST

```
10 POKE 23624,71: POKE 23693,71:CLS: LOAD ""CODE: LOAD ""SCREEN$
20 POKE 64017,31: POKE 64018,250
30 FOR I = 64031 TO 64045
40 READ A: POKE I, A
50 NEXT I
60 DATA 33,87,160,62,182,119,33,93,118,62,195,119,195,203,92
100 RANDOMIZE USR 64000
```

Том 3. Методы известных взломщиков компьютерных программ к ZX SPECTRUM.

Введение.

Эта книга написана для тех, кто внимательно изучил информацию первого и второго тома и теперь желают расширить свои знания путем исследования методов известных хаккеров.

Начало взлома компьютерных программ к "ZX SPECTRUM" относится приблизительно к 1986-1987 г.г. По всей видимости, это связано с тем, что фирмы-изготовители программного обеспечения с этого периода начали интенсивно защищать свои продукты от несанкционированного просмотра. Это, в свою очередь, было связано с существенными прорывами в технике программирования для даного компьютера и, кроме того, к такому поведению фирмы подтолкнуло широкое распространение самого компьютера.

Комментарий "ИНФОРКОМА"

Конечно же техника защиты программ, как и техника их взлома начали разрабатываться раньше. Можно сказать, что только в 1982 году выпускались программы без защиты, но и компьютеров в то время было еще очень немного.

В 1983 г. техника программирования была еще недостаточно развита для серьезных защит и, как правило, программы защищались только на уровне БЕЙСИК-загрузчика (совмещение цвета, внедрение управляющих кодов, искажение содержимого некоторых системных переменных, защита от MERGE).

1984 год стал годом разгула средств защиты. Появились такие варварские изобретения, как "стукалка", "спидлок", "лензлок". Вся мерзость их состояла в том, что не то, что скопировать программу было невозможно, ее и загрузить-то было трудно (в случае "лензлока" с загрузкой не было проблем, но были проблемы с запуском). Массовые жалобы покупателей и отказ от приобретения программ у фирм, запятнавших себя такими приемами стали коммерческой проблемой уже в 1985 году, когда убытки от средств защиты превысили убытки от пиратства.

Автор статьи относит начало появлений средств защиты к 1986 году и мы с ним согласны, если рассматривать квалифицированные средства защиты, построенные на изощренной логике программирования, а не на искажении параметров загрузки, что очень плохо отражается на самой загрузке.

Разумеется, техника взлома, техника защиты и техника программирования совершенствовались одновременно. В частности, если сравнить по графике игры 1986 года и 1989, то без сомнения в большинстве случаев Вы отдадите предпочтение последним. То же самое можно сказать и об уровне защищенности компьютерных программ.

Однако то же самое мы можем сказать и о технике взлома. В самом деле, фирмы совершенствуют свою защиту и, чтобы вскрыть ту или иную программу, приходится прикладывать максимум изобретательности.

Среди "хаккеров" наибольшую известность приобрел BILL GILBERT. Можно с уверенностью сказать, что программы взломанные им есть у каждого обладателя

компьютера "ZX SPECTRUM". Достаточно сложно судить о том, что заставило этого человека заниматься такой деятельностью, однако судя по распространению программ вскрытых Билом Гилбертом можно предположить следующее.

В нашу страну программы попадают уже пройдя через руки этого "хаккера", а поскольку наибольший приток программ сюда наблюдается из Польши, то не исключено, что он проживает именно там. Не исключено так же, что человека с таким именем не существует, а хаккер в своей деятельности использует псевдоним.

однако это всего лишь гипотеза, и если кому-либо известна достоверная информация о работе "хаккера", то я надеюсь, он поделится ей на страницах "ZX-РЕВЮ".

Комментарий "ИНФОРКОМа"

Не имея достоверной информации о Б. Гилберте, мы должны, тем не менее, отметить тот факт, что вопросы пиратского распространения копий программного обеспечения несколько лет назад широко обсуждались на страницах зарубежных журналов, посвященных Синклер-совместимым машинам. Сейчас острота этой проблемы спала, поскольку лидирующее место на рынке заняли другие компьютеры.

Исследования западных журналистов показали, что центром международного пиратства является Голландия, где по-видимому и следовало бы искать Б. Гилберта. Оттуда поток разделялся на три ветви. За океан в Южную Америку, в частности в Бразилию, где ситуация очень напоминала нашу - очень дорогие компьютеры при очень дешевых программах и при полном отсутствии элементарной защиты авторских прав. Второй поток - на юг в Израиль, далее куда угодно, и третий поток - на Восточную Европу, в частности в Польшу, Венгрию и Югославию, а из Польши - к нам.

Как показали журналистские расследования, эта деятельность - отнюдь не невинные развлечения энтузиаста "хаккера". В нее были вовлечены колоссальные средства, измерявшиеся миллионами долларов. Достаточно сказать, что во многих случаях пиратский тираж программ из Голландии в десяток раз превосходил основной тираж в Англии. Организация имела мощные сети и неоднократно начинала продавать вскрытые программы за несколько месяцев до того, как в Англии выходил первый защищенный оригинал.

Достаточно сложно выделить "хаккеров", которые бы занимали второе место после Билла Гилберта. Фактом является то, что никому не удалось еще подняться на его уровень. Тем не менее, в коллекции автора второе место занимает "PEGAS SOFTWARE". Несмотря на то, что никакой информации о месте нахождения данной "компании" нет. Это не помешает нам разобрать приемы, используемые ими в работе.

Ступенькой ниже идут отдельные программы, в которых окозалось сообщение о том, что их кто-то взломал. К таким хаккерам мы можем отнести "ROBI CRACKING SERVISE", "TOMI & DALI" и др.

Автор считает своим приятным долгом выразить благодарность тов. Кириллову С.В. из пос. Мурмаши Мурманской обл., предоставившему очень интересную информацию из журнала "Bajtek", касавшуюся данного вопроса.

ГЛАВА 1.

Техника защиты совершенствуется постоянно. Поэтому Глава 4 первого тома к моменту прочтения Вами изложенного в ней материала по всей видимости безвозвратно устарела. Именно по этой причине мы будем говорить о современных методах защиты как можно более часто, стараясь, чтобы изложенная информация сохраняла свою актуальность.

Итак, сегодня мы рассмотрим некоторые приемы, которые были взяты на вооружение фирмами-изготовителями программного обеспечения. Для начала исследуем очень любопытный прием, разработанный фирмой ULTIMATE.

Вы знаете, что "SPECTRUM" имеет "встроенные часы", роль которых выполняет системная переменная FRAMES. Ее значение увеличивается каждую 1/50 сек. Таким образом, мы можем контролировать процессы, протекающие строго определенное время. Именно на этом принципе основана одна из защит фирмы ULTIMATE (изготовитель

программ ATIC ATAC, SABRE WOLF, KNIGHTLORE, ALIEN 6, PENTAGRAM, NIGHT SHADE и мн. др.)

Поскольку при загрузке программ система прерываний отключается, то мы можем контролировать - осуществлялось или нет вмешательство в данную программу, следя за состоянием системной переменной FRAMES. В самом деле, для осуществления вмешательства необходима остановка программы, которая, что вполне естественно, приведет к несовпадению контрольных значений в системной переменнной FRAMES.

Рассмотрим, как это осуществляется на практике. Обычно, после достаточно простого загрузчика на БЕЙСИКе на ленте идет сама программа, после которой следуют три небольших блока, которые собственно и осуществляют защиту. Первый - это однобайтовый код инструкции JP(HL), осуществляющий старт необходимых процедур, второй блок программной защиты состоит из нескольких байтов - он осуществляет декодирование всей программы. (Вопросы кодирования и декодирования программ будут подробно рассмотрены в следующем разделе), и, наконец, третий блок имеет длину два байта и загружается в область 23627, т.е. в системную переменную FRAMES. Загружаемые значения представляют собой контрольную сумму, по величине которой и осуществляется проверка: останавливалась исходная программа или нет. Обычно этим занимается специальная процедура в машинных кодах, которая при обнаружении взлома обнуляет память компьютера.

Данная защита не получила широкого распространения и обычно ее можно обнаружить лишь в программах фирмы ULTIMATE. Это объясняется тем, что ее можно достаточно легко свести на нет, если придерживаться специального принципа при взломе.

Вмешательство в программу такого типа весьма просто. Достаточно загрузить все блоки, за исключением последнего, а после осуществления просмотра или проведения в ней определенных изменений (например, вписание необходимых POKES) достаточно ввести:

LOAD "" CODE:RANDOMIZE USR ADRESS(24064)

Эту команду необходимо обязательно ввести с клавиатуры одной строкой, разделяя инструкции двоеточием, чтобы вложиться в интервал времени соответствующий новому значению системной переменной FRAMES.

Поскольку в большинстве вышеприведенных программ фирмы ULTIMATE адресом старта является 24064, то именно он приведен в качестве примера. Однако, в каждом конкретном случае значение может быть другим. Необходимо каждый раз перед вводом этой серии команд уточнять его и вносить необходимые изменения.

После такого пояснения мы надеемся, что у читателя не возникнет затруднения при вводе POKES для изменения игр фирмы ULTIMATE.

(Продолжение следует)

МОНИТОР 48 - новые возможности.

Читателям, которые работают с машинными кодами Z80, наверняка приходилось использовать пакет программ фирмы PICTURESQUE "EDITAS/MONITOR". Существуют две версии данного пакета программ, рассчитанные на работу в компьютерах с оперативной памятью 16 и 48 килобайт. Они имеют соответственно названия MONITOR 16 и MONITOR 48. Естественно, программа, рассчитанная на 16 килобайтную машину, подходит к компьютеру, имеющему объем оперативной памяти 48 Кбайт, поэтому в отечественных моделях возможно и желательно использование обеих типов данных программ, т.к. вследствие своего расположения в низких адресах оперативной памяти программа MONITOR 16 может применяться для дисассемблирования блоков, занимающих верхние области памяти.

Однако было бы по меньшей мере наивно предполагать, что в программе MONITOR 48 существенным отличием является лишь новое расположение в оперативной памяти компьютера. Эта версия является одним из лучших отладчиков для программ в нашинных кодах. Она, безусловно, имеет меньше режимов работы, чем версии программы MONS из

пакета DEVPAC, однако это компенсировано разработчиками введением дополнительного режима, обеспечивающего работу программы через отладчик. Рассмотрим его более подробно.

Этот режим вызывается после нажатия клавиши T <ENTER> и служит для пошаговой отладки программы, написанной в машинном коде. Назовем этот режим режимом трассирования (T - TRACE).

После этого Вы попадаете в основное меню данного режима. На экране Вы можете видеть следующую картину (см. рис. 1):

- два дисассемблированных оператора текущего программного адреса;
- содержимое всех регистров микропроцессора в т.ч. и альтернативных;
- состояние всех флагов регистра F, в том числе и альтернативного.
- содержимое нескольких ячеек вершины машинного стека;
- содержимое определенных ячеек памяти, которые заранее выбраны пользователем.

Следует отметить, что содержимое основных регистров процессора отделено от содержимого альтернативных регистров. Кроме этого, напротив каждой основной регистровой пары приведено содержимое ячеек памяти, на которые указывает содержимое данной пары.

```
STEP
2D28     AF       LD C,A
IR        3F50             STACK
           SZ H PNC        0009
A'F'    >0001000100        000D
B'C'     1721              10ED
D'E'     369B              F39C
H'L'     2758             *3365
           SZ H PNC
AF       0F01010100
BC       F70F (BC) C3 45 F1 1A CD
DE       61C4 (DE) 00 00 0F F7 00
HL       2D2B (HL) FD 21 3A 5C AF

IX       03D4
IY       5C3A
SP       E8D6
PC       2D28
M000    F3 AF 11 FF FF C3 CB 11 2A
```

Экран режима "T" программы "MONITOR 48" при входе в него сразу после начала работы отладчика.

Рис. 1

Все это стандартная информация, которую можно получить и в MONS за исключением, пожалуй, достаточно интересной информации о содержимом стека. Фактически этот режим создан исключительно для изучения работы программ в машинных кодах. Здесь можно изучать пошаговое выполнение программы, поскольку после нажатия клавиши ENTER происходит выполнение текущей команды.

Теперь настало время рассмотреть более подробно работу программы в данном режиме. Я предполагаю, что читатель, заинтересовавшийся этим материалом уже имеет необходимое представление о машинных кодах и поэтому не считаю нужным давать здесь какую-либо информацию о процессоре Z80. Если Вы хотите получить эту информацию, рекомендую трехтомник "ИНФОРКОМа".

Итак, вы уже вероятно поняли, что информация, появившаяся на экране после перехода в режим "T" отражает текущее состояние микропроцессорной системы. Естественно, она будет изменяться после выполнения компьютером очередной команды. Сверху вы можете видеть две дисассемблированные команды, начиная с адреса, указанного в регистре PC микропроцессора. Вы сможете изменить этот адрес для того, чтобы

осуществить выполнение программы оттуда, откуда вам надо. Но об этой чуть позже, а теперь краткая сводка команд, использующихся в этом режиме.

S - установка программного счетчика (регистра PC) на заданный шестнадцатиричный адрес.

M - распечатка в нижней строке экрана содержимого ячеек памяти, начиная со значения, заданного после M. Всего можно увидеть 9 значений. В момент первого запуска данного режима программа автоматически устанавливает значение M, указывающее на нулевую ячейку памяти.

Курсорные клавиши - изменяют местоположение стрелки от одной регистровой пары к другой. Таким образом, можно указать на любой регистр микропроцессора, за исключением регистра программного адреса PC. Это необходимо, чтобы изменять содержимое данных регистров, а как Вам уже известно, содержимое регистра PC изменяется с помощью команды S.

1 - Если Вы нажмете на 1, то в строке ввода информации (самая нижняя строка экрана, она находится под строкой информации о содержимом памяти) появится содержимое текущей регистровой пары микропроцессора, на которую указывает стрелка-курсор. Если вы хотите изменить содержимое данной регистровой пары, то наберите шестнадцатиричное число, которое бы Вы хотели видеть вместо текущего значения. Если же Вы не хотите ничего изменять, то не набирая шестнадцатиричного числа нажмите ENTER. Кроме этого, для выхода из режима можно использовать команду X.

X - если Вы набираете какую-либо команду и увидели, что набираете ее неправильно, то ввиду того, что в программе MONITOR 48 отсутствует функция DELETE. Вам необходимо использовать функцию X. После нажатия этой клавиши вы возвращаетесь в исходный режим, который был до начала набора данной команды, примечательно, что эта клавиша используется и для выхода из режима "T" в основной режим MONITORа. Поэтому будьте осторожны и внимательны при использовании этой функций в данном режиме. Но в крайнем случае не отчаивайтесь: ничто не помешает Вам вновь нажать "T" и продолжить работу.

ENTER - нажатие этой клавиши приводит к выполнению компьютером текущей программной команды. Если вы несколько раз нажали клавишу ENTER, перед этим установив программный счетчик, то можете включить два вспомогательных режима.

R - после нажатия этой клавиши и ENTER справа вверху появляется надпись SCIP TO RET. Надо признаться, что этот режим еще недостаточно хорошо исследован, однако есть предположение, что по этой команде начинается исполнение программы и при обнаружении ближайшей команды RET осуществляется возврат в MONITOR 48. Предлагаю Вам самостоятельно проверить правильность этой гипотезы и подтвердить ее или опровергнуть.

Следующий режим исследован наиболее тщательно и, на взгляд автора, является тем преимущественным фактором, который вызывает предпочтение MONITORа 48 другим отладчикам. Это так называемый режим B.

B - нажатие этой клавиши и ENTER приводит к появлению слева вверху надписи BREAKPOINT, а повторное нажатие ENTER приводит к включению режима. После B можно набрать шестнадцатиричный адрес, с которого Вы бы хотели осуществить включение режима.

Данный режим работы служит для относительно медленного контроля работы исходной программы в машинных кодах с возможностью в любое время прервать выполнение программы нажатием клавиши BREAK. Причем в данном случае медленная работа программы Вам на руку, поскольку программы в машинных кодах в нормальном режиме выполняются очень быстро, а здесь происходит замедление в 100, а может и более раз. К тому же несомненное удобство представляет возможность остановки программы с

возвратом в отладчик. Вам не придется кусать себе локти в случае зависания отлаживаемой программы

Однако, есть несколько нюансов, которые необходимо учитывать при работе. Во-первых, если Вы наберете B0000, то, быть может, Вам вновь придется загружать программу с внешнего носителя. Это необходимо учитывать в том случае, если проверяемая Вами процедура использует в своей работе верхнюю область памяти (нельзя допускать каких-либо изменений в области, где хранится программа MONITOR 48).

Во-вторых, есть одна особенность, без учета которой бывает достаточно сложно включить режим "B". В частности, он не всегда начинает работать, если вы включаете его сразу после изменения содержимого программного счетчика, используя команду S. Если это происходит, то необходимо перед нажатием "B" "прогнать" несколько шагов программы нажатием клавиши ENTER.

Безусловно данная статья не в состоянии охватить полностью всех возможностей этого режима. Однако этого вполне достаточно, чтобы дать толчок для ваших исследований, уважаемые читатели. Мы надеемся, что открыв что-нибудь новое, Вы поделитесь своей информацией на страницах "ZX-РЕВЮ". Мы также надеемся, что быть может откликнется человек, имеющий необходимое описание, сделанное фирмой производителем данной программы, чтобы мы с вами могли более полно использовать все возможности великолепной программы-отладчика машинных кодов MONITOR 48.

Алексеев А. Г.

ПРОФЕССИОНАЛЬНЫЙ ПОДХОД

"ДЕБЮТ ПРОГРАММЫ"

Продолжая разговор о повседневных задачах, начатый в одной из номеров "ZX-РЕВЮ", следует поподробнее остановиться вот на чем. Несмотря на то, что Ваши программы различаются в зависимости от конкретной задачи, практически во всех программах встречается повторяющиеся моменты. Это может быть введение русского шрифта, титульная заставка с фамилией автора (Вашей), блок кодов "ON ERROR GO TO" с механизмом его инициирования, который описан в предыдущей статье (см. ZX-РЕВЮ 5-6 за 1992 г. стр. 113), какое-то стандартное начало программы. Неплохо также предусмотреть элементарный сервис для себя, в частности упрощение сохранения программы на ленте в процессе написания и отладки. То есть речь идет о "дебюте" программы. Наверное, у многих програмистов есть свой "дебют" или стандартное начало своих программ, а также одинаковые приемы, используемые практически во всех или в большинстве своих программ.

Предлагаю вниманию начинающих достаточно универсальный "дебют" программы. Я сам уже долгое время пользуюсь этим "дебютом" при разработке своих программ и оказывается, что при составлении новой программы требуется минимум переделок. Новую программу я всегда начинаю с загрузки "дебюта", затем набиваю остальные строки программы, не отвлекаясь на "второстепенные" задачи по формированию символьного набора и т.п.

Итак, сначала текст заготовки программы, назовем ее "PROG", затем - комментарии.

```
0 REM (машинные коды)
1 GO TO 100
2 RANDOMIZE 2: GO SUB 10:CLEAR 49999: LOAD "prog"COOE 50000,1000
3 GO SUB 8: PRINT . . . (вывод на экран заставки)
4 GO SUB 20: GO TO 0
5 GO SUB 9: SAVE "PROG"LINE 2: SAVE "prog"CODE 50000,1000
6 VERIFY "PROG": VERIFY "prog" CODE: PRINT INK 9; TAB 20; "O.K." : BEEP 0.5,32: GO TO 5
7 POKE 23675,208: POKE 23676,95: RETURN : REM UDG
8 POKE 23606,208: POKE 23607,91: RETURN : REM RUS
9 POKE 23606,0: POKE 23607,60: RETURN : REM LAT
10 POKE 24746, PEEK 23670: POKE 24749,PEEK 23671: RANDOMIZE USR 24696: RETURN :REM ON ERROR
      GO TO
20 PRINT #0; TAB 5;"НАЖМИТЕ ЛЮБУЮ КЛАВИШУ": PAUSE 0: INPUT;
22 IF INKEY$="q" THEM GO TO 9999
24 RETURN
100 RANDOMIZE 3: GO SUB 10: GO SUB 7: GO SUB 8: BORDER 1: PAPER 0: INK 6: BRIGHT 0: CLS
1000 PAUSE 50: GO SUB 20
1010 PRINT AT 12, 10;"ЗВУК 1":RANDOMIZE USR 24769
1020 PAUSE 50: GO SUB 20 1030 PRINT AT 12,10;"ЗВУК 2":RANDOMIZE USR 24801
9999 BORDER 7: PAPER 7: INK 0: BRIGHT 0: INVERSE 0
```

В нулевой строке программы расположены блоки кодов, необходимые для ее работы.

О формировании нулевой строки мы подробно поговорим позже. А сейчас - о том, что же это за блоки кодов.

1. Символьный набор расположен с адреса 23760. Длина - 768 байт (по адрес 24537). Системная переменная CHARS при этом равна: 23760-256=23504. Символьный набор может быть любым, по вашему вкусу. Я использую русско-латинский символьный набор в кодах ASCII КОИ-7 "НС". Подробно об этом - в статье о русификации программы "МАСТЕРФАЙЛ-09" в ZX-РЕВЮ 1-2 за этот год, стр. 31. Там же программа по формированию такого символьного набора.

2. Символы UDG графики расположены с адреса 24528. Длина -168 байт (по адрес 24695). Этот набор может быть совершенно различным в зависимости от Ваших требований. Попутно замечу, что часто могут требоваться изображения стрелок влево, вправо, вниз (вверх - уже есть в символьном наборе), элемента для изображения рамок, волнистые линии и т.д. Все зависит от требований Вашей программы. Поэтому я не буду специально останавливаться на этом.

3. Блок кодов "ON ERROR GO TO" расположен с адреса 24696. Длина - 73 байта. Подробно работа этого блока была изложена в предыдущей статье в ZX-РЕВЮ N 5-6 за этот год на стр.113. Номер Бейсик-строки для перехода задается в ячейках 24746,24749.

4. SOUND 1 - так назовем коротенький блок в машинных кодах, выдающий звуковой сигнал, который может использоваться для индикации какого-либо положительного эффекта в программе (это может быть правильный ответ на вопрос или находка "клада" и т.п.). SOUND 1 расположен с адреса 24759 (по адрес 24800). Длина - 32 байта. Вызов - RANDOMIZE USR 24769.

5. SOUND 2 - аналогично предыдущему пункту - звуковой сигнал, индицирующий отрицательный эффект в программе, т.е. неправильный ответ на вопрос, ошибка в действиях игравшего и т.п. SOUND 2 расположен с адреса 24801. Длина - 32 байта. Вызов - RANDOMIZE USR 24801.

Для получения блоков кодов SOUND 1 и SOUND 2 приведена программа в конце этой статьи, а более подробно о них будет рассказано в следующем выпуске "РЕВЮ".

Конечно, это не означает, что Вы должны копировать этот порядок. Вам просто надо определиться со своими требованиями и добавить свои блоки кодов или изменить имеющиеся.

Теперь о других строках программы.

Автостарт программы происходит со строки 2. Строки 2, 3, 4 – выполняют те действия, которые необходимо сделать, чтобы обеспечить старт программы "с нуля", после включения компьютера. Иными словами, это "горячий старт". Сюда входит следующее. Предположим, что для работы Вашей программы требуется загрузка некоторого блока в машинных кодах. Пусть это будет блок "prog"CODE 50000,1000.

Тогда предварительно надо переустановить RAMTOP для резервирования места под блок кодов командой CLEAR (ADDR-1), где ADDR - адрес загрузки.

Далее (строка 3) может идти вывод сообщения об авторах программы, дате ее последней коррекции или какие-либо рекламные комментарии, а также краткие сообщения по работе программы. (Перед тем, как это сделать, скорее всего придется включить русский символьный набор.) Если Вам недостаточно одной строки для этого, то используйте GO SUB ...

Строка 4 передает управление на начало программы, обеспечивая "холодный" старт программы. В процессе отладки Вы будете пользоваться "холодным" стартом программы, подавая команду RUN. При этом нет необходимости опять загружать блок машинных кодов. В том случае, если у вас нет блока машинных кодов, который необходим для работы Вашей программы и "холодный" и "горячий" старт - это одно и то же, то исключите строки 1 и 2, а в строке 4 GO TO 0 исправьте на GO TO 100. 100 - это адрес начала непосредственного выполнения программы.

Зато, если вы в дальнейшем захотите усовершенствовать свою программу, например, "озвучить" ее с помощью музыкального редактора "WHAM", то Вы всегда сможете восстановить строки 1 и 2 для загрузки скомпилированная кодов мелодий.

Строки 5 и 6 обеспечивают вам возможность периодического сохранения программы (вместе с блоком кодов, если это необходимо) на ленте в процессе написания и отладки. Для этого надо сделать RUN 5.

Строка 7 - переключатель символов UDG-графики. Она включает набор символов UDG, размещенный в нулевой строке.

Строка 8 - включение символьного набора, который расположен в нулевой строке.

Строка 9 - выключение символьного набора нулевой строки (т.е. включение символьного набора из ПЗУ).

Строка 10 - подготовка и активизация блока кодов "ON ERROR GO TO", подробно об этом было изложено в предыдущей статье в ZX-РЕВЮ 5 -6 за этот год.

В строках 20, 30, 40, ... по 99) - располагаются недлинные подпрограммы, которые часто используются на протяжении всей программы, типа вывода таблички "нажмите любую клавишу" или последовательности операторов BEEP, играющих мелодию из нескольких нот и т.п. В общем, их можно было бы располагать и в другом месте, например в строках 6000 или 9000, но я использую именно строки с 20 по 99 просто потому, что короче набирать GO SUB 20, чем GO SUB 8000.

Сама программа начинается со строки 100. Причем в строках 100 999 располагаются следующие фрагменты. Это установка исходных параметров, когда происходит переключение блока UDG-графики на тот, который загружен с ленты, включение нового символьного набора, устанавливаются необходимые цвета PAPER, INK, BORDER и т.п. Могут задаваться массивы для переменных, присваиваются численные значения переменным и т. д.

Со строки 200 может быть расположено главное меню программы. Подробно об универсальном варианте такого меню будет рассказано в следующей статье. Это меню, в свою очередь, может передавать управление на другие "подменю", располагаемые со строк 300, 400 ... , при помощи которых управление передается на непосредственное выполнение различных фрагментов программы. Последние располагаются, начиная со строк 1000, 2000 и т.д. (Для примера, в строкая с 1000 демонстрируются звуки SOUND 1 и SOUND 2).

Остановка программы, в случае использования блока кодов "ON ERROR GO TO", возможна только благодаря "жучку" в строке 22. Нажав клавишу "Q", когда появляется табличка: "нажмите любую клавишу", переходим на строку 9999. Эта строка обеспечивает восстановление белого цвета PAPER и черного INK при остановке программы для удобства редактирования. Добавив в строку 9999: GO SUB 9, можно включать опять символьный набор Спектрума, но если Вы используете русско-латинский символьный набор "НС" в кодах КОИ-7, то это переключение не требуется. Наоборот, в этом случае вы будете одинаково хорошо видеть текст программы, печатаемый английскими буквани и текстовые сообщения в операторах PRINT, печатаемые русскими буквами. Надо только для удобочитаемости программы имена переменных набирать, используя режим CAPS LOCK, тогда они будут печататься английскими буквами. (Впрочем, может быть Вам больше понравится набирать их русскими словами? Дело вкуса.)

Теперь поговорим подробнее о нулевой строке. Вспомним, как располагается в памяти строка Бейсика. На начало первой строки программы указывает системная переменная PROG (2 байта по адресу 23655). Обычно это 23755. 2 байта занимает номер строки, (младший и старший байты здесь расположены не как обычно, сначала – старший, затем - младший) затем 2 байта занимает длина строки (длина текста плюс 1 байт для кода 13, завершающего строку), затем идет текст программы на Бейсике, затем код 13 - ENTER - конец строки.

Если набрать 1 REM , а после REM, скажем, 5 пробелов, то после ввода в память строка будет иметь вид:

```
23755 0 номер строки: 1
23756 1
23757 7 длина строки: 7
23756 0
23759 234 REM
23760 32
23761 32
23762 32
23763 32
23764 32
23765 13 <ENTER>
```

Область за REM начинается с адреса 23760 и заканчивается адресом 23764 (5 байт).

Теперь на это место можно загрузить блок кодов (конечно, если его длина не превышает 5 байт, иначе Бейсик-программа будет запорчена). Так мы делали, располагая в начальной строке программы блок кодов "ON ERROR GO TO" (см. предыдущую статью в ZX-РЕВЮ 5-6 на стр. 113). Там для этого надо было 73 байта памяти и после REM мы набирали 73 пробела.

Для размещения тех кодов, о которых говорилось выше, надо зарезервировать следующий объем памяти:

Символьный набор:	768 байт
UDG-графика:	168 байт
ON ERROR GO TO:	73 байта
SOUND 1:	32 байта
SOUND 2:	32 байта
Всего:	1073 байта

То есть после REM должно быть набрано 1073 пробела или любых других символов. Если Вы попытаетесь вручную набрать такое количество пробелов, то это займет уйму времени, к тому же, при наборе, с определенного момента времени начнет раздаваться предупредительный звуковой сигнал, еще более замедляющий работу. То есть, набрать вручную такое количество пробелов практически невозможно.

В этом случае могла бы помочь опять же программа "SUPERCODE". Под номером 84 в ней находится блок кодов "EXPAND REM" (или "REM FILL" в "SUPERCODE 2"). Этот блок кодов должен расширять область REM на величину до 9999 байт. К сожалению, этот блок кодов имеет программные ошибки, из-за чего он не работает так, как должен. Поэтому я для создания REM-области пользуюсь своей программой в машинных кодах, которую хочу предложить вниманию читателей.

Для тех, кто интересуется машинными кодами, приводится текст программы с комментариями. Если Вас это не интересует, то пропустите описание работы блока кодов.

Программу назовем так же, как и в "SUPERCODE" - "REM FILL".

Блок кодов "REM FILL"

Этот блок кодов можно загружать в любое место памяти, для примера, он загружен в буфер принтера, в адрес 23296 (#5B00). В отличие от программы N84 в "SUPERCODE", здесь длина получаемой REM-области не ограничена 9999 байтами, а ограничена только памятью компьютера, отведенной под Бейсик-программу (системной переменной RAMTOP). Кроме того, сам блок кодов почти вдвое короче, чем в "SUPERCODE". Единственное ограничение на работу блока кодов - это то, что получаемая REM-область должна быть не менее двух байт (что вряд ли кому-то может придти в голову).

Теперь - блок кодов "REM FILL", затем - комментарии.

```
5B00 016400          LD BC,#0064        (1)
5B03 2A4B5C          LD HL,(#5C4B)      (2)
5B06 09              ADD HL,BC          (3)
5B07 224B5C          LD (#5C4B),HL      (4)
5B0A 2A555C          LD HL,(#5C55)      (5)
5B0D 09              ADD HL,BC
5B0E 22555C          LD (#5C55),HL
5B11 2A595C          LD HL,(#5C59)      (6)
5B14 09              ADD HL,BC
5B15 22595C          LD (#5C59),HL
5B18 2A5B5C          LD HL,(#5C5B)      (7)
5B1B 09              ADD HL,BC
5B1C 225B5C          LD (#5C5B),HL
5B1F 8A5D5C          LD HL,(#5C5D)      (8)
5B22 09              ADD HL, BC
5B23 225D5C          LD (#5C5D),HL
5B26 2A615C          LD HL,(#5C61)      (9)
```

```
5B29 09             ADD HL,BC
5B2A 22615C         LD (#5C61),HL
5B2D 2A635C         LD HL,(#5063)      (10)
5B30 09             ADD HL,BC
5B31 22635C         LD (#5C63),HL
5B34 DD2A535C       LD IX,(#5C53)      (11)
5B38 DDE5           PUSH IX
5B3A DD360000       LD (IX+0),#00      (12)
5B3E DD360100       LD (IX+1),#00
5B42 DD6603         LD H,(IX+3)        (13)
5B45 DD6E02         LD L,(IX+2)
5B48 E5             PUSH HL            (14)
5B49 09             ADD HL,BC          (15)
5B4A DD7403         LD (IX+3),H        (16)
5B4D DD7502         LD (IX+2),L
5B Э0 D1            POP DE             (17)
5B51 13             INC DE
5B52 13             INC DE
5B53 13             INC DE
5B54 E1             POP HL             (18)
5BS5 19             ADD HL,DE          (19)
5B56 C5             PUSH BC            (20)
5B57 C5             PUSH BC
5B58 EB             EX DE,HL           (21)
5B59 2A655C         LD HL,(#5C65)      (22)
5B5C E5             PUSH HL            (23)
5B5D ED52           SBC HL,DE          (24)
5B5F E5             PUSH HL            (25)
5B60 C1             POP BC
5B61 D1             POP DE             (26)
5B62 E1             POP HL
5B63 19             ADD HL,DE          (27)
5B64 22655C         LD (#5C65),HL      (28)
5B67 EB             EX DE,HL           (29)
5B68 2B             DEC HL             (30)
5B69 1B             DEC DE             (31)
5B6A EDBB           LDDR               (32)
5B6C 23             INC HL             (33)
5B6D 3600           LD (HL),#00        (34)
5B6F E5             PUSH HL            (35)
5B70 D1             POP DE
5B71 13             INC DE             (36)
5B72 C1             POP BC             (37)
5B73 0B             DEC BC
5B74 EDB0           LDIR               (38)
5B76 00             NOP                (39)
5B77 210100         LD HL,#0001        (40)
5B7A CD6E19         CALL #196E
5B7D E5             PUSH HL
5B7E 211027         LD HL,#2710
5B81 CD6E19         CALL #196E
5B84 D1             POP DE
5B65 CDE519         CALL #19E5
5B88 C9             RET
```

Комментарии к программе.

Сначала (1) в регистр BC заносится длина будущей REM-области - это та величина, на которую удлинится начальная строка Бейсик-программы, или иными словами, это то число байтов, которое добавится между текстом начальной строки и кодом 13 - <ENTER>, завершающим строку. (Для примера задано 100 байтов - то есть #0064.)

Далее следует ряд действий по изменению значений системных переменных. На величину добавляемых байтов должна увеличиться значения следующих системных

переменных:

VARS - адреса переменных Бейсика (23627 или #5C4B)
NXTLIN - адрес следующей строки Бейсик-программы (23637 или #5C55)
E_LINE - адрес выведенной команды (23641 или #5C59)
K_CUR - адрес курсора (23643 или #5C5B)
CH_ADD - адрес следующего интерпретируемого символа (23645 или #5C5D)
WORK_SP - адрес временной рабочей области (23649 или #5C61)
STK_BOT - адрес дна программируемого стека (23651 или #5C63)
STK_END - адрес начала резервной области памяти (23653 или #5C65)

Все эти системные переменные имеют длину по два байта и изменяются следующим образом. Сначала (2) в регистр HL заносится содержимое системной переменной (VARS), затем (3) к нему прибавляется число добавочных байтов из регистра BC и затем результат сложения (4) из регистра HL пересылается назад в память компьютера, в ячейку этой системной переменной.

Аналогичные действия (5)-(10) производятся и с остальными перечисленными системными переменными, кроме STK END. О ней - чуть позже.

Далее (11) в регистр IX заносится значение системной переменной PROG (23635). Теперь в IX -23755. Это число нам еще понадобится для дальнейших рассчетов, поэтому сохраним его на стеке.

Записывая 0 по адресу (PROG) и (PKOG)+1 (12) изменяем номер начальной строки программы, теперь это 0.

Затем (13) в регистры H и L побайтно заносятся числа из адресов (PROG)+2 и (PROG)+3. Теперь в регистре HL - длина начальной строки Бейсик-программы. Запомним это число на стеке (14) для дальнейших действий. Далее содержимое HL также увеличивается на величину добавляемых байтов и затем (16) новое значение длины строки из регистров H и L побайтно записывается в память.

Для дальнейших рассчетов нам надо получить адрес символа <ENTER> - конца строки. Начиная с этого места вся дальнейшая область Бейсик-системы должна быть отодвинута на величину добавляемых байтов. Для этого снимаем со стека в регистр DE бывшую длину начальной строки и увеличиваем ее на три (два байта - номер строки плюс два байта - длина строки и минус один байт - символ <ENTER>; итого 3 байта). Затем (18) снимаем со стека значение PROG и добавляем к нему (19) полученное число из регистра DE. Теперь в регистре HL - адрес символа <ENTER>. Теперь, перед дальнейшими расчетами, сохраним на стеке (20) длину добавочных байт.

Затем (21) переписываем содержимое HL в DE (это адрес символа <ENTER>), а в регистр HL загружаем (22) значение системной переменной STK END число из ячейки 23653. Бейсик-система заканчивается адресом (STK END)-1, а с адреса (STK END) начинается свободная область памяти. Сохраним (23) эту величину на стеке. Теперь (24) из HL вычтем DE. Получим длину блока, который надо отодвинуть для размещения добавочных байтов. Перепишем (25) это число через стек из HL в BC. далее (26) снимем со стека значение STK END в регистр DE и число добавочных байтов в регистр HL. сложив их (27), получим (в регистре HL) новое значение STK END. Теперь (28) его можно занести в память в таблицу системных переменных.

Далее (29), поменяв между собой содержимое регистров HL и DE, получим в HL - бывшее значение STK END, а в DE - новое значение STK END. Уменьшив HL на единицу (30), получим последний адрес Бейсик-системы, которую надо отодвинуть (путем переброски) на величину добавляемых байтов. Адрес "места назначения" перебрасываемого блока кодов получится (31) в результате уменьшения на единицу содержимого DE. В BC к этому моменту находится длина перебрасываемого блока - начиная с символа <ENTER> - конца начальной строки до величины STK END.

Для переброски блока кодов используется (32) команда LDDR, а не LDIR, так как LDDR начинает переброску со старших адресов и заканчивает младшими, что сохраняет массив

кодов при частичном наложении (см. "Программирование в машинных кодах"; М.:"ИНФОРКОМ", 1990,1992, 271 стр.).

К моменту завершения переброски массива, в регистре HL будет 23759 - адрес последнего "непереброшенного" байта. Увеличив (33) это значение на единицу, получим адрес первого "добавочного" байта. Иными словами - это адрес первого байта за REM (то есть 23760).

Для того, чтобы полученную REM-область очистить от прежних кодов - остатков переброшенной Бейсик-программы (заполнить например нулями), воспользуемся командой LDIR. Для этого сделаем следующее. Запишем (34) в первую ячейку за REM - ноль. Затем (35) перенесем через стек содержимое HL в регистр DE и увеличим (36) на единицу содержимое DE. Теперь (37) снимем со стека число добавочных байтов в регистр BC и уменьшим это число на единицу (так как в одну ячейку мы уже занесли ноль). При выполнении LDIR (38) происходит следующее. Из ячейки 23760 ноль переписывается в 23761, затем из 23761 - в 23762 и так далее, пока не заполнится нулями вся добавочная REM-область кодов.

Вместо нуля можно задать любой другой код, изменив числовой параметр в команде (34). Задав, например, 32 (#20), вся добавленная REM-область заполнится пробелами, а задав 137 - графическими символами типа "шахматная клетка".

На этом работа по формированию REM-области заканчивается. Если на месте команды NOP (39) поставить RET, то произойдет возврат из машинного кода в вызывающую Бейсик-программу. Но в данном варианте происходит переход на дальнейшее выполнение программы в машинных кодах (40). Этот небольшой фрагмент служит для удаления строк с 1 до 10000 вызывающей Бейсик-программы. Строка с REM-областью имеет теперь номер 0 и не удаляется. Этот блок кодов (40) взят из ZX-РЕВЮ N3 за 1991 г. со стр. 50, поэтому я не останавливаюсь на нем подробно. В результате действия этого блока кодов в памяти компьютера останется только нулевая строка с REM-областью. Ее можно теперь записать на магнитофон или соединить с необходимой Бейсик-программой при помощи MERGE.

Для того, чтобы получить блок кодов "REM FILL", наберите программу на Бейсике:

```
10 LET N=23296: LET S=0
20 FOR X=N TO N+136
30 READ Y
40 POKE X.Y
50 LET S=S+Y
60 NEXT X
70 IF S<> 12985 THEN PRINT FLASH 1;"ERROR": STOP
80 SAVE "REM FILL"CODE 23396,137
90 STOP
100 DATA 1,2,0,42,75,92,9,34,75,92,42,85,92,9,34,85,92,42,89,92,9,34,89,92
110 DATA 42,91,92,9,34,91,92,42,93,92,9,34,93,92,42,97,92,9,34,97,92,42,99,
      92,9,34,99,92
120 DATA 221,42,83,92,221,229,221,54,0,0,221,54,1,0,221,102,3,221,110,2,229,
      9,221,116,3,221,117,2
130 DATA 209,19,19,19,225,25,197,197,235,42,101,92,229,237,82,229
140 DATA 193,209,225,25,34,101,92,235,43,27,237,184,35,54
150 DATA 0,229,209,19,193,11,237,176,0
160 DATA 33,1,0,205,110,25,229,33,16,39,205,110,25,209,205,229,25,201
```

Если Вы все набрали правильно, то программа сформирует и выдаст для записи на магнитофон блок кодов под именем "REM FILL". В строке 150 первое значение DATA определяет код символа, которым будет заполнена REM-область.

Бейсиковая часть программы "REM FILL" выглядит следующим образом:

```
1 REM 10 LOAD "REM FILL"CODE 23296
20 INPUT "No. of extra bytes:"; n
30 RANDOMIZE n: POKE 23297, PEEK 23670: POKE 23298, PEEK 23671
40 CLEAR : RANDOMIZE USR 23296
```

Программа стартует сначала, загружая блок кодов "REM FILL". Затем в строке 20 запрашивается, на сколько байтов надо увеличить начальную строку. (При этом в строке 1 после REM уже может находиться какая-либо информация, например, Ваша фамилия и телефон. Добавочные байты будут расположены после этой информации.) Строка 30 преобразует число добавочных байтов в двухбайтную форму и записывает полученное значение в ячейки 23297, 23296, подготавливая данные для блока кодов. Затем стартует блок кодов "REM FILL". В результате, после сообщения 0 OK. будет сформирована строка с заданным числом байтов после REM, номер этой строки станет 0, а все остальные строки Бейсик-программы будут уничтожены.

А теперь вернемся к нашему дебюту программы "PROG". В этом случае для формирования нулевой строки надо на запрос о числе добавочных байтов ответить: 1073. После того, как нулевая строка необходимой длины будет сформирована, наберите (или догрузите) текст программы "PROG". (Но не запускайте ее. Кроме этого, во избежание ошибок, подставьте RETURN: в начало строки 10. Это отключит пока блок кодов "ON ERROR GO TO" до тех пор, пока не будет набита, отлажена и работать без ошибок вся программа. Только после этого можно начинать вторую часть работы - отладку программы с блоком "ON ERROR GO TO", удалив RETURN из начала строки 10.)

Теперь можно загрузить в REM-область заранее подготовленные символьный набор, блок UDG и другие блоки кодов:

```
LOAD "CHR"CODE 23760,768
LOAD "UDG"CODE 24528,168
LOAD "ON ERR"CODE 24696,73
LOAD "SOUND 1"CODE 24769,32
LOAD "SOUND 2"CODE 24801,32
```

Если в начальной строке вы расположили после REM свою фамилию, то адреса загрузки кодовых блоков должны быть пересчитаны. Они должны увеличиться на столько, сколько занимает текст с Вашей фамилией.

Что касается блоков кодов "SOUND 1" и "SOUND 2", то они сформированы при помощи программы "SPECSOUND" фирмы "OZ SOFTWARE". Те читатели, которые не имеют этой программы, могут получить указанные блоки (для записи на магнитофон), набрав и запустив следующую Бейсик-программу:

```
10 LET N=23296: LET S=0
20 FOR X=N TO N+63
30 READ Y
40 POKE X,Y
50 LET S=S+Y
60 NEXT X
70 IF S<>6020 THEN PRINT FLASH 1;"ERROR": STOP
80 PRINT AT 10,12;"SOUND 1": RAHDOMIZE USR 23296: PAUSE 50: PRINT AT 10,18;"2": RANDOMIZE
     USR 23328: PAUSE 100
90 CLS : SAVE "SOUND 1"CODE 23296,32: SAVE "SOUND 2"CODE 23328,32
100 DATA 14,1,6,5,33,224,1,197,17,70,0,229,205,181,3,225,17,40,0,237,
     82,193,16,239,62,2,12,65,184,32,227,201
110 DATA 14,1,6,100,33,200,0,197,17,10,0,229,205,181,3,225,17,2,0,237,
     90,193,16,239,62,2,12,65,184,32,227,201
```

После старта, если Вы все набрали правильно, программа сформирует блоки кодов и выдаст их для записи на магнитофон.

В одной из следующих статей я подробно остановлюсь на работе этих блоков кодов, а также предложу усовершенствованный вариант программы "SPECSOUND", переведенный на русский язык.

И, в заключение, еще один момент. Если Вы размещаете блоки кодов в нулевой строке Бейсик программы, то отсутствует жесткая привязка этих кодов к конкретным адресам памяти компьютера, а есть только привязка к началу Бейсик-программы -

системной переменной PROG. Если Вы работаете с магнитофоном и у Вас нет планов обзаводиться дисководом, то все будет нормально. Если же Вы решите в будущем адаптировать Вашу программу для работы с дисковой операционной системой TR-DOS, то вы должны знать, что для работы этой системы в памяти компьютера выделяется дополнительно 112 байт ОЗУ, в которых хранятся новые системные переменные, связанные с работой TR-DOS. Эти дополнительные байты расположены непосредственно перед Бейсик-программой, поэтому последняя отодвинута на 112 байт в памяти компьютера и системная переменная PROG имеет значение не 23755, а 23867. Так что при работе с дисковой операционной системой все обращения к кодам нулевой строки должны быть смещены на 112 байт.

Аналогичные сюрпризы могут возникнуть не только при работе с дисковой операционной системой. В частности - если Вы пользуетесь ПЗУ "TURBO-90", которое имеет встроенный МОНИТОР и возможность загрузки программ с магнитофона с удвоенной скоростью. В некоторых режимах работы этого ПЗУ происходит сдвижка Бейсик-программы в область более старших адресов.

Ключей к избавлению от этих "подводных камней" является тот факт, что куда бы ни была сдвинута Бейсик-программа, ее место всегда указывается в системной переменной PROG. Исходя из этого, начало символьного набора в нулевой строке можно представить как (PROG)+5. При этом CHARS=(PROG)+ 5-256=(PROG)-251, начало UDG-графики (PROG)+5+768=(PROG)+773, начало блока "ON ERROR GO TO" будет: (PROG)+5+768+168=(PROG)+941 и т.д.

Теперь надо изменить некоторые строки нашего "дебюта":

```
7 RANDOMIZE PEEK 23635+256*PEEK 23636+773:POKE 33675, PEEK 23670: POKE 23676, PEEK 23671:
      RETURN : REM UDG
8 RANDOMIZE PEEK 23635+256*PEEK 23636-251: POKE 23606, PEEK 23670: POKE 23607, PEEK 23671:
      RETURN : REM RUS
10 POKE PEEK 23635+256*PEEK 23636+993, PEEK 23670: POKE PEEK 23635+256*PEEK 23636+994,PEEK
      23671: RANDOMIZE USR (PEEK 23635+256*PEEK 23636+941): RETURN
```

Строку 9 изменять не надо. Вызов звуков SOUND 1 и SOUND 2 может осуществляться через GO SUB. Определим для этого, скажем, строки 11 и 12:

```
11 RANDOMIZE USR (PEEK 23635+256*PEEK 23636+1014): RETURN
12 RANDOMIZE USR (PEEK 23635+256*PEEK 23636+1046): RETURN
```

Тогда вызов SOUND 1 будет GO SUB 11, а SOUND 2 GO SUB 12.

Этот вариант обращения к кодам нулевой строки предпочтительнее, он гораздо более универсален, так как не требует какой-либо переделки для адаптации программы под БЕТА-ДИСК интерфейс, да и в других случаях не принесет Вам сюрпризов. Но, поскольку ничего в природе не дается даром, то такой вариант имеет и свой недостаток - он медленнее работает. Работая над программой, оцените сами, существенно ли это замедление для Вас, и решите, какой вариант Вам более подходит.

*   *   *

Рассматривая дебют программы "PROG", я упомянул о том, что со строк 200, 300... могут быть расположены меню программы. В следующей статье мы подробно рассмотрим возможный вариант такого меню. Оно достаточно эффектно выглядит на экране и легко переналаживается на разные режимы работы.

УНИВЕРСАЛЬНОЕ МЕНЮ

Итак, Вы получили весь необходимый аппарат, который можете использовать, начиная дописывать к "дебюту" текст Вашей программы. У Вас уже есть переключатели

UDG-графики, русского и латинского символьного наборов. Вы можете заблокировать остановку программы и имеете кое-какие удобства для пользования - автоматизм сохранения программы на ленте.

Итак, предположим, что, загрузив "дебют", Вы исправили строку 3 программы. Теперь она вывела на экран название вашей будушей программы, Вашу фамилию, дату написания, и внизу экрана появилась табличка: "нажмите любую клавишу". Что же дальше?

Как правило, ни одна игра не начинается сразу. Необходимо сначала задать управление, выбрать уровень сложности и т.д. Для этого можно использовать цифровые и буквенные клавиши. Обычно это делается по принципу:

```
100 INPUT "Введите уровень сложности (1-5) : "; n
```

или так:

```
100 PRINT "Бесконечные жизни (Y/N)?": PAUSE 0
110 IF INKEY$ "y" OR INKEY$ "Y" THEN ...
```

Однако наиболее профессиональным является вариант меню. Меню - это перечень возможных вариантов ответа, например:

```
KEMPSTON
SINCLAIR
PROTEK
CURSOR
KEYBOARD
```

Здесь каждая строка – это пункт меню. Один из пунктов меню всегда выделен (яркость, цвет, инверсия). При этом переход от одного пункта к другому происходит при нажатии клавишей "ВВЕРХ" или "ВНИЗ", а выбор выделенного пункта осуществляется и клавишами "ENTER" или "0" ("ОГОНЬ"). Таким образом, в различных ситуациях в зависимости от требуемого запроса будет меняться текст меню, а управление будет всегда одинаковым и может осуществляться, например, только курсор джойстиком, то есть без помощи клавиатуры. Я хочу предложить вниманию читателей достаточно отработанный вариант меню, которое применяю в своих программах. Оно хорошо смотрится на экране и легко перенастраивается под разные конкретные варианты. Чтобы пример не был абстрактным, предположим, что мы разрабатываем программу для обучения детей устному счету. В этой программе, (назовем ее "PRIM") компьютер задает примеры на сложение, вычитание и т.д. и контролирует результат игравшего. Кроме того, в игре предусмотрим режим, когда играющий сможет сам задавать примеры компьютеру. При этом компьютер превращается в калькулятор (забегая вперед, скажу, что моя дочь, например, с удовольствием проверяет таким способом домашнее задание). Этот режим делает игру более разнообразной, и к тому же, психологически это как бы ставит компьютер и играющего в равные условия. Зададим еще режим "КОНЕЦ РАБОТЫ", когда компьютер поблагодарит Вас за уделенное ему время.

Главное меню этой программы может выглядеть следующим образом:

```
ПРИМЕРЫ ЗАДАЕТ КОМПЬЮТЕР
ПРИМЕРЫ ЗАДАЕТ ИГРАЮЩИЙ
КОНЕЦ РАБОТЫ
```

Если Вы выберете первый пункт меню, то будет сделан переход на следующее меню, которое позволит выбрать тип решаемых примеров:

```
СЛОЖЕНИЕ
СЛОЖЕНИЕ И ВЫЧИТАНИЕ
УМНОЖЕНИЕ
УМНОЖЕНИЕ И ДЕЛЕНИЕ
ВСЕ ЧЕТЫРЕ ДЕЙСТВИЯ
ТАБЛИЦА УМНОЖЕНИЯ
КОНЕЦ РАБОТЫ
```

Алгоритм работы программы должен быть следующим. При выборе одного из первых пяти пунктов, уровень сложности должен постепенно увеличиваться при правильных ответах

и уменьшаться при определенном количестве неверных ответов, а при выборе шестого пункта уровень сложности на протяжении всей игры должен оставаться постоянным. При выборе последнего пункта происходит возврат в предыдуюее, главное меню.

Продолжая придерживаться структуры "дебюта" "PROG", определим для главного меню строки с 200 по 299, а для второго меню - строки с 300 по 399. При этом строки начиная с 30 (до 40), являются подпрограммой, выполняющей действия, непосредственно связанные с работой меню.

Итак, текст программы, затем - дальнейшие комментарии.

```
4 GO TO 100
30 RANDOMIZE 3: GO SUB 10: BORDER 1: PAPER 7: INK 0: CLS
31 GO SUB 8: FOR M= 1 TO NN: READ M$: PRINT AT ,(Y0+(M-1)*DY),X0;M$: NEXT M: PRINT AT Y1,3;
     INK 1; INVERSE 1; "SPACE,DOWN,UP,ENTER,BREAK"
32 IF MM<1 THEN LET MM=NN
33 IF MM>NN THEN LET MM=1
34 PRINT AT (Y0+(MM-1)*DY),X0-DX; PAPER 2; INK 6; BRIGHT 1; OVER 1;S$: BEEP .03,2*MM+10
35 PAUSE 0: LET I=(INKEY$=" " OR INKEY$:"6" OR CODE INKEY$=10)+(INKEY$="7" OR CODE
     INKEY$=11)*2*(INKEY$="0" OR CODE INKEY$=12 OR CODE INKEY$=13)*3: GO TO (35+I)
37 PRINT AT (Y0+(MM-1)*DY),X0-DX; OVER 1;S$: LET MM=MM-2*I+3:GO TO 32
38 BEEP .1,36: RETURN 190 LET MM=1
200 RESTORE 200: LET NN=3: LET Y0=8: LET X0 = 4: LET DY=2: LET DX=2: LET S$="
                    ": LET Y1=18: GO SUB 30:GO TO 200+MM*10
210 LET MM= GO TO 300 220 GO TO 8000
230 GO TO 9000
299 DATA "ПРИМЕРЫ ЗАДАЕТ КОМПЬЮТЕР", "ПРИМЕРЫ ЗАДАЕТ ИГРАЮЩИЙ","      КОНЕЦ РАБОТЫ"
300 RESTORE 300: LET NN = 7: LET Y0=7: LET X0 = 6: LET DY=1: LET DX=2: LET S$="
                ": LET Y1 = 19: GO SUB 30: GO TO 300+MM*10
360 GO TO 1000
370 LET MM=3: GO TO 200
399 DATA "СЛОЖЕНИЕ", "СЛОЖЕНИЕ И ВЫЧИТАНИЕ","УМНОЖЕНИЕ","УМНОЖЕНИЕ И ДЕЛЕНИЕ", "ВСЕ ЧЕТЫРЕ
     ДЕЙСТВИЯ","ТАБЛИЦА УМНОЖЕНИЯ","КОНЕЦ РАБОТЫ"
```

Текстовые сообшения в программе напечатаны по-русски. Они будут так выглядеть на Вашем экране, если вы включите русско-латинский символьный набор командой GO SUB 8. Однако следует учесть, что имена переменный вводятся латинскими буквами, для этого надо включить режим CAPS LOCK.

В программе меню используются следующие переменные:

NN - число пунктов в меню.

X0 - левая граница (по горизонтали) текста меню.

Y0 - верхняя граница (по вертикали) текста меню.

DX - отступ выделяющей строки влево по отношению к тексту меню.

DY - шаг строк меню по вертикали.

Y1 - вертикальная координата вспомогательной строки с указанием управляющих клавишей.

MM - указатель меню.

Указатель меню MM - это основной параметр меню. Это выходной параметр, получаемый в результате работы меню. Это также и входной параметр, определяющий выделенный пункт при старте меню.

Перед тем, как начнется выполнение меню со строки 200 или 300, необходимым предварительным условием является задание указателя меню MM. Это сделано для того, чтобы можно было, обращаясь к меню из разных мест программы, выделять тот пункт, выбор которого в данный момент наиболее вероятен. Это создает повышенное удобство в работе и придает "профессионализм" программе. Поясню это на нашем примере. Так, при старте программы выделяется первый пункт главного меню: "ПРИМЕРЫ ЗАДАЕТ КОМПЬЮТЕР" (в строке 190 задается MM=1) потому, что именно этот режим наиболее вероятен для выбора. Далее, нажав ENTER, мы перейдем ко второму меню. Если теперь выбрать "КОНЕЦ РАБОТЫ", то на экране появится главное меню с выделенным третьим пунктом, а не первый, как было при старте программы. Повторное нажатие ENTER, для

подтверждения, и без лишних хлопот для оператора программа завершена. То есть в каждой конкретном случае обрашения к меню, может быть выделен тот пункт, который наиболее вероятно будет выбран оператором, то есть программа отличаясь "деликатностью", как бы "подыгрывает" оператору, облегчая его действия.

Учитывая вышесказанное, процедура вызова меню всегда одинакова и имеет вид:

```
LET MM=...: GO TO ...
```

Это, например, строка 210 (вызов второго меню) или 370 (вызов главного меню). Кроме того, это строка 190 (это тоже вызов главного меню; здесь нет необходимости указывать GO TO 200, так как следующей выполняемой строкой и так является строка 200).

Теперь конкретно о главном меню. Строка 200 задает начальные параметры: число пунктов в главном меню NN=3. Левая и верхняя границы текста меню X0 и Y0 - Вы можете подбирать их экспериментально, добиваясь симметричного расположения текста меню на экране. То же относится и к величине DX и количеству пробелов в переменной S$. Что касается параметра DY, то если меню состоит из небольшого количества пунктов (2... 5), то лучше смотрится вариант, когда текст написан через строчку, то есть DY=2. Если число пунктов в меню большое, то строки следуют вплотную друг к другу (DY =1).

Далее в строке 200 идет выполнение подпрограммы меню: GO SUB 30. Такое выделение процедур выполнения меню в подпрограмму позволяет значительно сэкономить память компьютера в программах, где требуется сложная сеть многоступенчатых, взаимосвязанных между собой меню. Если в вашей программе меню используется только один раз, то, в принципе, можно было бы раскрыть подпрограмму, расположив весь текст меню в строках с 200. Но я не рекомендую это делать, вполне вероятно, что в будущем Вы захотите усовершенствовать программу, введя какие-то дополнительные режимы, для которых потребуются новые меню. Тогда Вы оцените такой подход.

Теперь подробно рассмотрим подпрограмму меню.

Строка 30 задает возврат при ошибке на перезапуск программы сначала при нажатии клавиши BREAK (со строки 3; подробно об этом говорилось в ZX-РЕВЮ). Далее идет задание цветов BORDER, PAPER, INK и очистка экрана.

Строка 31 выводит на экран текст меню, а также вспомогательный текст с указанием управляющих клавишей. При этом команда GO SUB 8 включает русско-латинский символьный набор.

Строки 32 и 33 проверяют указатель меню MM на допустимые пределы и "зацикливает" переход от одного пункта меню к другому, то есть, если нажать клавишу "ВНИЗ", то, дойдя до последнего пункта, при следующем нажатии будет сделан скачок к первому пункту. Аналогично - при движении вверх. Вы можете ликвидировать "зацикливание", сделав замену:

```
32 IF MM<1 THEN LET MM=1
33 IF MM>MM THEN LET MM=NN
```

Строка 34 выделяет пункт меню, определенный указателем MM. При этом звуковой сигнал как бы подтверждает этот факт, а тон сигнала ориентировочно указывает на предлагаемый выбор.

Строка 35 - режим ожидания нажатия клавиши. При этом в зависимости от нажатой клавиши меняется параметр I.

I=1, если нажаты клавиши "SPACE" или "6" или "ВНИЗ" (CAPS SHIFT+6)

I=2, если нажаты клавиши "7" Или "ВВЕРХ" (CAPS SHIFT+7)

I=3, если нажаты клавиши "0" или "DELETE" (CAPS SHIFT+0) или "ENTER"

I=0, если нажата любая другая клавиша.

Далее в строке 35 следует переход на строку 35+I. То есть, в последнем случае (I=0), будет сделан переход на эту же строку 35 и ожидание нового нажатия клавиши, в случае I=1 - переход на строку 36, но так как ее нет, то будет сделан переход на ближайшую следующую строку, то есть 37, как и в случае I = 2.

В строке 37 (если I=1 или 2) выделенный пункт меню становится невыделенным, указатель меню MM в зависимости от величины I либо увеличивается на единицу, либо

уменьшается на единицу и работа меню повторяется со строки 32, то есть проверка ММ на допустимые пределы, выделение нового пункта меню и ожидание нажатия клавиши.

В случае, когда I=3, строка 38 осуществляет выход из меню и действия, связанные с этим, например, звуковой сигнал. Экспериментируя с меню, попробуйте изменить строку 38:

```
38 FOR M=1 TO NN: PRINT PAPER 8: INK 8. BRIGHT 8; OVER (M=MM);FLASH (M=MM);AT (Y0+(M-
    1)*DY),X0-DX;S$: NEXT M: PRINT AT Y1, 0;"          (32 пробела)          ": BEEP
    .1,36: PAUSE 10: RETURN
```

Такой вариант интереснее смотрится на экране. Попробуйте сами придумать какие-нибудь другие варианты.

По команде RETURN в конце строки 38 закончится работа меню (GO SUB 30) и мы возвращаемся на строку 200, где после GO SUB 30 следует переход на строку, номер которой вычисляется выражением 200+MM*10, то есть для трех возможных вариантов главного меню: строки 210,220,230.

Как уже говорилось выше, в том случае, если выбран первый пункт главного меню, то вызывается второе меню (строка 210). В двух других случаях управление передается непосредственно на соответствующие фрагменты программы (строки 220, 230).

Второе меню (строки 300...399) полностью идентично первому. Отличие состоит только в числе пунктов, работа его абсолютно такая же, как и главного меню.

Строка 300, в случае нажатия "ENTER" или "0", адресует программу к строкам, начиная с 310 по 370, в зависимости от величины указателя ММ. Отсюда уже выполняются переходы на дальнейшие фрагменты программы. Так как строки 310, 320, 330, 340 и 350 - отсутствуют, то в случае ММ=1...6 будет переход на строку 360, а отсюда должен быть организован переход на начало основной части программы - запрос уровня сложности игры.

Теперь отдельно обсудим момент, связанный с работой блока кодов "ON ERROR GO TO". Я использую следующий прием. Если в тот момент, когда на экране находится меню, нажать клавишу "BREAK", то программа перезапускается со строки 3 (см. строку 30), выводя на экран вступительную заставку с названием, фамилией и т.д. Почему именно со строки 3 - см. статью о структуре программы и дебюте "PROG". При этом обеспечивается "полухолодный" (или "полугорячий") старт программы, в отличие от полностью "холодного" старта со строки 2, где может быть размещена загрузка каких-либо кодовых кусков, например, скомпилированных кодов мелодии для озвучивания игры, приготовленных при помощи музыкального редактора "WHAM". Это могут быть и какие-нибудь другие усовершенствования, поэтому лучше не занимать строку 2.

Так выглядит работа универсального меню.

Для тех читателей, кого больше интересует конкретная игровая программа, чем отдельные куски, из которых она собрана, немного ниже приводится окончание программы "PRIM" для обучения детей устному счету. Это та часть программы, которая не имеет отношения к работе меню. Поэтому она выделена в отдельный листинг.

А теперь небольшое отступление от темы и возврат к теме предыдущей номера. А именно, с чего практически начинать написание (набивание) новой программы.

Вообще, я делаю так. Дебют "PROG" набран и хранится отдельно. Начиная воплощать в жизнь какую-то идею, я прежде всего загружаю его. Затем решаю, понадобится ли для этой программы меню. Если понадобится, то догружаю при помощи "MERGE" текст меню, который также набран и хранится отдельно (строки с 30 по 38). При этом вовсе не обязательно определяться с необходимостью меню в начале написания программы. Например вы пишете какую-то простенькую вспомогательную программу для своих целей, и вот, когда задача успешно выполнена, Вы решаете, что хорошо было бы добавить в программу какие-то новые режимы. Вот тут то Вы имеете возможность догрузить меню в вашу программу и с его помощью организовать новые режимы работы. Сами же отдельные фрагменты программы, тоже могут быть набраны и опробованы отдельно, так как для каждого блока программы определяются свои области, которые не должны перекрываться (со строк 1000, 2000 и т.д.). Смысл в том, что придерживаясь идеи "дебюта" и структуры программы, вы в любой момент можете усовершенствовать или видоизменить ее по

Вашему желанию. В этом и заключаются преимущества структурного программирования, о котором уже говорилось на страницах "ZX-РЕВЮ".

Прежде чем предложить листинг программы "PRIM", еще одно отступление. Точнее это не отступление, а целое маленькое исследование, но без него непонятен будет смысл некоторых изменений, которым подверглись знакомые уже Вам (по дебюту "PROG") фрагменты и приемы.

Наберите для этого простенькую программку:

```
10 RANDOMIZE
20 LET X=RND*100
30 LET Y=RND*100
40 PLOT X,Y
50 GO TO 20
```

Запустите ее. То, что вы видите на экране, является результатом работы функции случайной величины RND. Вы видите, что картина подобна той, которую рисует на асфальте начинающийся дождь. То есть, распределение в достаточной степени случайно.

Для того, чтобы задать начальный параметр для функции случайной величины, служит оператор RANDOMIZE n, где n - числовой параметр. Если этот параметр равен нулю (RANDOMIZE 0 или просто RANDOMIZE), то начальный параметром для RND служит значение системной переменной FRAMES - счетчика кадров. Так обеспечивается практически случайное число.

Измените теперь строку 50:

```
50 GO TO 10
```

И запустите программу RUN. то, что Вы видите сейчас, вовсе не похоже на дождь. То есть для нормального функционирования RND нельзя включать RANDOMIZE внутрь цикла работы RND. А то как бы все время функция RND запускается заново а первое ее значение совсем не случайно, оно определяется системной переменной FRAMES.

Продолжаем наше исследование. Измените строку 10:

```
10 RANDOMIZE 13345
```

(можете подставить любое число) и запустите программу. Теперь вообще нет никакого "дождя", все "капли" попадают в одну и ту же единственную точку на экране. Сейчас мы каждый раз в начале цикла задаем для RND одно и то же число и поэтому каждый раз получаем одинаковый результат. Все эти детали нужны вот для чего. В основе программы "PRIM" лежит использование функции RND для задания примеров ("ПРИМЕРЫ ЗАДАЕТ КОМПЬЮТЕР"). А сплошь и рядом в тексте программы будут встречаться: RANDOMIZE или RANDOMIZE USR ... это вызов SOUND или SOUND 2, подготовка двухбайтных данных для блока кодов "ON ERROR GO TO", запуск самого этого блока. Естественно, все эти RANDOMIZE попадают внутрь цикла работы RND, нарушая его функционирование.

Измените строку 10:

```
10 RANDOMIZE USR 124
```

По адресу 124 в ПЗУ находится команда RET, то есть сразу выполняется возврат, ничего не делая, но имитируется выполнение программы в машинных кодах. Запустите программу. На экране по-прежнему одна единственная точка. Как же выполнить программу в машинных кодах, не прибегая к помощи RANDOMIZE? Это сделать можно, используя вместо RANDOMIZE USR ... другой вариант:

```
10 LET S=USR 124
```

Примечание "ИНФОРКОМа"

Возможно использование также RESTORE USR, PRINT USR, PLOT USR и т.д. Кратко принцип формулируется следущим образом: "Если в программе используется генерация случайных чисел, то использовать RANDOMIZE для запуска машинного кода нельзя".

Подробности вы можете посмотреть в нашей новой книге "Элементарная графика". М:."ИНФОРКОМ", 1993, 207 стр. (это первый том готовящегося четырехтомника).

Измените строку 10 и запустите программу. Вот теперь на экране опять "дождь". RND работает правильно.

Кроме того, придется отказаться от услуг RANDOMIZE по превращению чисел в двухбайтную форму. Для этого сделаем новую подпрограмму в строке 15. Число, подлежащее преобразованию в двухбайтную форму обозначим W. Тогда старший (HI) и младший (LO) байты будут вычисляться в строке 15:

```
15 LET HI=INT (W/256): LET LO=W-HI*256: RETURN: REM 2-BYTE CONVERTER
```

Учитывая все вышесказанное, начальные служебные строки (с 7-й) будут теперь такими:

```
7 LET W=PEEK 23635+256*PEEK 23636+773: GO SUB 15: POKE 23675,LO: POKE 23676,HI: RETURN: REM
     UDG
8 LET W=PEEK 23635+256*PEEK 23636-251: GO SUB 15: POKE 23606,LO: POKE 23607,HI: RETURN: REM
     РУС
9 POKE 25606,0: POKE 23607,60: RETURN: REM ЛАТ
10 GO SUB 15: POKE PEEK 23635+256*PEEK 23636+993, LO: POKE PEEK 23635+256*PEEK 23636+994,HI:
     LET S=USR (PEEK 23635+256*PEEK 23636+941): RETURN
11 LET S = USR (PEEK 23635+256*PEEK 23636+1014): RETURN: REM SODND 1
12 LET S = USR (PEEK 23635+256*PEEK 23636+1046): RETURN: REM SOUND 2
15 см. выше
```

Для инициирования блока кодов "ON ERROR GO TO" например в строке 30 вместо:

```
RANDOMIZE 3: GO SUB 10
```

теперь используется другая конструкция:

```
LET W=3: GO SUB 10
```

а в начало строки 10 теперь подставлено GO SUB 15, где число и преобразуется в двухбайтную форму.

Если Вы догружаете программу меню с магнитофона, то не забудьте изменить начало строки 30.

Программа "PRIM"

Теперь, наконец, переходим к тексту остальных строк программы "PRIM". Комментарии - потом.

```
0 REM коды
3 LET W=3: GO SUB 10: GO SUB 7: GO SUB 8: BORDER 1: PAPER 0: INK 6: BRIGHT 0: CLS : PRINT
     BRIGHT 1;AT 2,2: "ПРОГРАММА ДЛЯ ОБУЧЕНИЯ СЧЕТУ": PAPER 2; INK 7; INVERSE 1; AT 7,9: "
     ";AT 8,9; "    P R I M    ";AT 9,9;"               "; PAPER 0: INK 2: INVERSE 0:AT
     14,6:"АВТОР: АЛЕКСЕЕВ А.Г."
4 PRINT INK 4: BRIGHT 1; AT 17,12;"01,06,92": BEEP .1,26: BEEP .1,20: GO SUB 20: GO TO 100
5 GO SUB 9: SAVE "PRIM" LINE 2: VERIFY "PRIM"
6 GO TO 5
7
.
. см. выше
.
15
20 INPUT ;: PRINT #0; "     НАЖМИТЕ ЛЮБУЮ КЛАВИШУ"
22 LET W=22: GO SUB 10: PAUSE 0
24 IF INKEY$="q" OR INKEY$="Q" THEN GO TO 9999
26 RETURN
30
.
. см. текст программы меню
.
38
40 LET T=0: RETURN
41 LET T=INT (RND*2): RETURN
42 LET T=2: RETURN
43 LET T=INT (RND*2+2): RETURN
44 LET T=INT (RND*4): RETURN
```

```
45 LET T=2: RETURN
50 LET W=50: GO SUB 10: INPUT O
52 IF O=0 THEN GO TO 4000
54 PRINT O;
56 LET W=3: GO SUB 10: RETURN
60 PRINT TAB 20; PAPER 2: INK 7; BRIGHT 1; " ОШИБКА "''
62 GO SUB 12
64 LET E=E+1: LET U=U+1
66 RETURN
70 PRINT TAB 20; PAPER 4; " ВЕРНО "''
72 GO SUB 11
74 LET R=R+1: LET I=I+1
76 RETURN
100 RANDOMIZE
190 LET MM=1 200
.
. см. текст программы меню
.
399
1000 BORDER 7: PAPER 7: INK 0; CLS
1010 LET W=1010: GO SUB 10: INPUT "ЗАДАЙТЕ УРОВЕНЬ : "; LINE L$
1020 IF L$="" OR L$<="0" THEN LET MM=2: GO TO 200
1030 LET L = VAL L$
1040 LET W=3: GO SUB 10: LET N=L: LET R = 0: LET E=0: LET I=1
1050 PRINT AT 11,0;"ДЛЯ ЗАВЕРШЕНИЯ РАБОТЫ ВВЕДИТЕ 0": BEEP .05,32
1100 PRINT AT 19,10; "УРОВЕНЬ ";INT L
1110 LET U=0
1120 FOR C=1 TO 5
1130 LET X=INT (RND*.7*L+.3*L+.5)
1140 LET Y=INT (RND*.7*L+.3*L+.5)
1150 PRINT AT 21,1; I;"."
1160 GO SUB 40+MM-1
1170 GO SUB 2000+100*T
1180 NEXT C
1200 PRINT '''''
1210 IF U=0 THEN PRINT AT 17,10:PAPER 6; INK 2; BRIGHT 1; " ОТЛИЧНО ": LET L =
      L+.2*L"(MM<>6)
1220 IF U=1 THEN PRINT AT 17,10;PAPER 5; INK 1;"  ХОРОШО  "
1230 IF U>=2 THEN PRINT AT 17,10; PAPER 1; INK 5; BRIGHT 1; " ПЛОХОВАТО ": LET
      L=L=.2*L*(MM<>6)
1240 GO TO 1100
2000 LET Z=X+Y
2010 PRINT AT 21,5;X;"+";Y;"=";
2020 GO SUB 50
2030 IF O<>Z THEN GO SUB 60: GO TO 2010
2040 GO TO 70
2100 LET Z=X+Y
2110 PRINT AT 21,5;Z;"-";X;"=";
2120 GO SUB 50
2130 IF O<>Y THEN GO SUB 60: GO TO 2110
2140 GO TO 70
2200 LET Z=X*Y
2210 PRINT AT 21,5;X;"x";Y;"=";
2220 GO SUB 50
2230 IF O<>Z THEN GO SUB 60: GO TO 2210
2240 GO TO 70
2300 LET Z=X*Y
2310 PRINT AT 21,5;Z;":";X;"=";
2320 GO SUB 50
2330 IF O<>Y THEN GO SUB 60: GO TO 2310
2340 GO TO 70
4000 LET W=3: GO SUB 10: BORDER 4: CLS : PRINT AT 1,5;"ТИП ПРИМЕРОВ: "; AT 3,6;
4010 RESTORE 300: FOR A=1 TO MM: READ A$: NEXT A: PRINT A$
4020 PRINT AT 7,5; "НАЧАЛЬНЫЙ УРОВЕНЬ : ";N;AT 9,5;"КОНЕЧНЫЙ УРОВЕНЬ : "; INT L
```

```
4030 PRINT AT 14,10; PAPER 4;" ВЕРНО "; PAPER 7;" : ";R;AT 15,10; PAPER 2; INK 7; BRIGHT 1;
     " ОШИБОК "; PAPER 7; INK 0; BRIGHT 0; " : "; E
4040 BEEP .1,36: BEEP .1,20: PAUSE 0
4050 LET MM=3: GO TO 200
8000 LET W=3: GO SUB 10: BORDER 7: PAPER 7: INK 0: CLS
8010 PRINT AT 11,0;"ДЛЯ ЗАВЕРШЕНИЯ РАБОТЫ ВВЕДИТЕ 0": BEEP .05,32
8020 PRINT AT 21,0;"ВВЕДИТЕ ПРИМЕР ( БЕЗ ЗНАКА = ) :"
8030 LET W=8030: GO SUB 10: INPUT LIKE X$: IF X$="0" THEN LET MM=1: GO TO 200
8040 LET X=VAL X$
8050 PRINT ''AT 21,0;X$;"=";X'''
8060 GO SUB 11: GO TO 8020
9000 LET W=9030: GO SUB 10: BORDER 3: PAPER 7: INK 0: CLS
9010 PRINT AT 11,10: INK 1;"ДО СВИДАНИЯ."'''" ПРИЯТНО БЫЛО С ВАМИ ПОРАБОТАТЬ!"
9020 BEEP .1,26: BEEP .1,20: PAUSE 0
9030 RANDOMIZE USR 0 9999 BORDER 7: PAPER 7: INK 0: BRIGHT 0: INVERSE 0: POKE 23658,8
```

Переменные, используемые в программе.
L$, L - уровень чисел для счета
N - начальный уровень
R - счетчик правильных ответов
E - счетчик ошибочных ответов
I - порядковый номер примера
U - счетчик ошибочных ответов в "столбике" из пяти примеров
C - параметр цикла, определяющий число примеров в "столбике"
X,Y,Z - операнды для задания примеров
T - тип примера
T=0 - сложение
T=1 - вычитание
T=2 - умножение
T=3 - деление
O - ответ, вводимый играющим.

В этой программе собраны все моменты, освещенные в предыдущих статьях цикла. Набирая и отлаживая программу, Вы еще раз остановитесь на них.

На подпрограммах в строках 40...66 мы остановимся по ходу работы основной части программы. Основная часть - режим задания примеров компьютером - начинается со строки 1000.

В строке 1010 запрашивается начальный уровень чисел для счета. Это величина, которую не могут превышать числа, например слагаемые при выполнении сложения.

Если не задавая никакого уровня нажать "ENTER", или задать уровень 0, то произойдет возврат в главное меню с выделением второй строки меню (установка режима задания примеров играющим). Чтобы реализовать возможность простого нажатия ENTER, в строке 1010 вместо INPUT L стоит INPUT LINE L$ и далее в строке 1030 числовой переменной L присваивается значение текстовой переменной L$.

Обратите внимание на применение блока кодов "ON ERROR GO TO". В строке 1010 перед INPUT стоит конструкция LET W=1010: GO SUB 10, которая в случае ошибки при вводе уровня возвратит программу на строку 1010 и повторит ввод. Такая ошибка может произойти в начале строки 1030, если, например, вместо числа, задать букву. Когда опасность такой ошибки миновала, корректно будет возвратить блок "ON ERROR GO TO" в исходное состояние (то есть при ошибке программа перезапускается со строки 3, как мы договаривались выше). Это происходит в строке 1040. Здесь также запоминается начальный уровень для подведения итогов в конце игры, обнуляются счетчики верных и ошибочных ответов, счетчику примеров присваивается номер 1.

Строка 1050 информирует играющего о том, как он может завершить работу программы.

Со строки 1100 начинается следующий этап в работе программы - задание "столбика" из пяти примеров. Вначале выводится текущее значение уровня чисел для счета. Затем в

строке 1110 обнуляется счетчик ошибок в одном столбике (U). Затем (в строке 1120) организуется цикл, определяющий пять примеров в "столбике".

В строках 1130 и 1140 задаются значения операндам, участвующим в примерах, при этом числа для счета получаются в диапазоне от 0.3*L до L. Такое ограничение позволяет, если уровень чисел для счета задан, например 100, избежать примеров типа: "1+2=".

Строка 1150 выводит на экран номер решаемого примера.

Подробнее разберемся со строкой 1160. Здесь, в зависимости от того, какой вариант из второго меню был выбран (указатель MM), выполняется соответствующая подпрограмма, задавая тип примера (T). Если было выбрано только сложение (MM=1), то выполняется GO SUB 40, то есть T становятся равным 0. Если сложение и вычитание (MM=2), то выполняется GO SUB 41, где T может принять одно из двух значение: 0 или 1 и т.д. В зависимости от величины T находится тип конкретного задаваемого примера:

T=0 - сложение
T=1 - вычитание
T=2 - умножение
T=3 - деление

Далее, строка 1170 в зависимости от величины T, вызывает выполнение соответственно одной из четырех подпрограмм из строк 2000, 2100, 2200 или 2300 для каждого типа примеров. Они идентичны, за исключением некоторых деталей. Для того, чтобы при вычитании не получалось отрицательных чисел, а при делении - дробных чисел, используется простой прием. Сначала подсчитывается результат, а пример строится, используя результат, как исходные данные.

Рассмотрим, для примера, подпрограмму со строки 2000. В строке 2000 осуществляется подсчет результата, в 2010 - вывод примера на экран. В следующей строке GO SUB 50 - это ввод играющим ответа и проверка его на ноль для выхода из игры. В этом случае будет переход на строку 4000 (см. строку 52). Так как при работе INPUT опять возможны всякие неприятности, в строке 50 активизируется блок "ON ERROR GO TO", заставляя программу возвращаться на эту строку, на ввод ответа. И опять корректно будет, завершая подпрограмму ввода ответа вернуть блок "ON ERROR GO TO" к переходу по ошибке на строку 3 перед тем, как выполнить RETURN в строке 56. Далее в строке 2030 идет анализ величины ответа, и, если ответ неверный, то выполнение подпрограммы GO SUB 60 - это реакция программы на ошибку и далее, возврат на повторение этого примера еще раз (со строки 2010).

В подпрограмме со строки 60 идет вывод таблички "ОШИБКА", затем звуковой сигнал SOUND 2, а также на единицу увеличивается счетчик ошибок (E) и счетчик ошибок в этом столбике (U).

Строка 2040 - выполнение подпрограммы при правильном ответе. Вообще-то наверное понятнее в строке 2040 было бы записать:

```
GO SUB 70: RETURN
```

То есть выполнение подпрограммы "ВЕРНО", затем возврат в вызывавшую подпрограмму - на строку 1180. Но результат одинаковый. В том варианте, который приведен в листинге, в вызывающую программу (на строку 1180) управление вернет оператор RETURN из строки 76. Подпрограмма со строки 70 - это вывод таблички "ВЕРНО", звуковой сигнал SOUND 1, затем увеличение на единицу счетчика правильных ответов (R) и увеличение на единицу сквозной нумерации примеров (I).

После возврата в вызывающую программу на строку 1180 происходит либо повтор цикла, то есть вывод нового примера из "столбика", либо "столбик" завершен и пора оценить промежуточный результат.

Строки 1210, 1220 и 1230 выполняют анализ числа неправильных ответов в одном столбике. Если неправильных ответов нет (строка 1210), то выводится оценка "отлично". Уровень чисел увеличивается на 20 процентов (но только в тех случаях, если не была выбрана таблица умножения, то есть если MM<>6), иначе уровень сложности не

увеличивается.

Если в столбике один неправильный ответ (строка 1220), то выводится сообщение "ХОРОШО", а уровень - не изменяется.

Если в столбике два или более неправильных ответов (строка 1230), то выводится табличка "ПЛОХОВАТО", а уровень чисел понижается на 20 процентов (если только это не режим таблицы умножения).

Когда будете придумывать текст сообщений компьютера оператору, следует воздержаться от резких и категоричных высказываний типа: "ОЧЕНЬ ПЛОХО" или "ВЫ НИЧЕГО НЕ ЗНАЕТЕ". Компьютер должен быть вежлив в общении с оператором, тем более, если оператор - ребенок. Даже если он совсем ничего не знает или невнимателен при вводе ответа, компьютер не ругает его нехорошими словами, а лишь слегка "недоумевает" по поводу неправильного результата. В этом случае у ребенка не возникает раздражения и не "отпадает охота", а появляется желание играть в эту игру и дальше. А ведь как раз это и нужно для того, чтобы развить способности к устному счету.

Кстати, вместо "ПЛОХОВАТО" можно выводить совсем нейтральное: "УПРОСТИМ ЗАДАЧУ".

Далее (строка 1240), работа программы возобновляется со строки 1100, то есть задание нового "столбика" примеров.

Если при выполнении подпрограммы ввода ответа (GO SUB 50) будет принят ноль (строка 52), то управление передается на строку 4000 - это подведение итогов работы, где на экран выводится следующая информация, число решенных примеров, их тип, начальный и конечный уровни решенных примеров, а также число верных и ошибочных ответов. Затем (строка 4050) вызывается главное меню, предлагая завершить программу (выделен третий пункт меню).

Строки с 8000 - эта часть программы выполняет режим калькулятора и может использоваться, например, при проверке учеником домашнего задания. В случае ошибки при вводе ответа, блок "ON ERROR GO TO" вернет программу на ввод примера, на строку 8030.

Строки с 9000 (по 9030) - финальная часть программы. Сюда можно добавить выполнение нехитрой мелодии (при помощи нескольких операторов BEEP), исполнение которой зациклено до нажатия на какую-нибудь клавишу. После нажатия на клавишу - рестарт компьютера. Попробуйте эту часть программы придумать сами.

Для того, чтобы в процессе отладки можно было останавливать программу, установлен "жучок" в строке 24, переводящий на строку 9999, которая обеспечит удобные для работы цвета бумаги, чернил и т.д., включение режима курсора [C] и остановку программы с сообщением 0 OK, так как это последняя строка программы. Практически, для остановки программы надо войти в любое меню, затем нажать BREAK (для перехода в титульную заставку), затем нажать "Q".

Остановку программы можно обеспечить по-другому, например непосредственно из режима меню.

Для этого измените строку 35:

```
35 PAUSE 0: LET I=(INKEY$=" " OR INKEY$="6" OR CODE INKEY$=10) +(INKEY$="7" OR CODE
     INKEY$=11)* 2+(INKEY$="0" OR CODE INKEY$=12 OR CODE INKEY$=13)*3+(INKEY$="q" OR
     INKEY$="Q")*4: GOTO (35+I)
```

И добавьте строку 39:

```
39 GO TO 9999
```

В общем, чем больше Вы будете экспериментировать, тем лучше. Это всегда полезнее, чем просто скопировать готовую программу. Важно понять подход, принцип, а уж как его развить и использовать дальше - это на Ваше усмотрение.

* * *

В тексте часто упоминаются блоки кодов SOUND 1 и SOUND 2. Они сформированы при помощи программы "SOUND". За основу этой программы была взята программа "SPECSOUND" фирмы "OZ SOFTWARE", которая была "доведена до ума" и переведена на русский язык. Следующая статья будет посвящена этой теме. Я подробно остановлюсь на работе этих блоков кодов, а также предложу читателям программу "SOUND".

Маленькие Хитрости

Признаться по совести, нам очень понравилась идея создания универсального программного "дебюта", предложенная автором предыдущей статьи. Конечно, "универсального" не в том смысле, что он должен быть один на всех и всех устраивать, а в том смысле, что каждый, кто программирует на "Спектруме", сможет сделать для себя собственную домашнюю заготовку и "подшивать" ее всякий раз, когда ему придет в голову написать какую-нибудь программу.

Идея "дебюта" - это не только программистская идея, ее ценность идет гораздо дальше. Из мемуаров известно, с каким трудом многие творческие личности берут себя по утрам за шиворот и тянут к письменному столу работать, а ноги упирается. Куда приятнее: подошел к столу, нажал пару кнопок и готово дело, дебют уже сделан, можно с чистой совестью отправляться обедать, а потом и вовсе отдыхать.

Многие писатели практикуют даже такой прием: отходя ко сну, они бросают своих героев на самом интересном месте, обрывая фразу на полуслове. А на следующий день спокойно с утра добьют своих героев, похоронят, если получится, и таким жизнерадостным дебютом начинают новый день в самой работоспособном настроении.

Вот и мы прочитали статью уважаемого автора и захотелось "плодотворную дебютную идею" (как говаривал высокочтимый нами Остап Ибрагимович) как-то развить и упрочить. Захотелось и свой вклад в большое дело внести. Попробовали и так и этак, ничего в голову не идет - надо все-таки программировать, а без этого ну просто никуда.

И тут нас осенило: ну зачем нам программировать!? Ведь в компьютере уже есть куча программ в ПЗУ. Может быть ими и воспользуемся? Зачем велосипед изобретать?

Взялись за дело. Все ПЗУ прошарили, все обыскали, пачку бумаги испачкали - но нет там ничего такого, чтоб компьютер сам за нас программу написал. Эх, лучше было бы это время на "ЭЛИТУ" потратить. И вдруг...

Самое интересное всегда происходит неожиданно. Вдруг по адресу 196ЕН (6510 - десятиричное) мы нашли процедуру, которая выдает адрес начала любой БЕЙСИК-строки в оперативной памяти, особенно если вы предварительно зашлете номер этой строки в регистровую пару HL процессора. После того, как процедура отработает, она оставит искомый адрес в той же регистровой nape HL.

НУ вот, решили мы, половина дела сделана. Адрес начала любой строки компьютер найдет сам, осталось только, чтобы он сам и строку туда записал. Но не тут то было - ничего для этого в ПЗУ нет. Тут что-то Клайв Синклер недодумал, за что только его лордом сделали?

Но мы не отчаялись и нашли еще одну интересную процедуру, которая начинается с адреса 19Е5Н (6629 DEC). Эта процедура способна уничтожить все, что находится в БЕЙСИК-программе между адресом, который установлен в регистровой паре DE, и адресом, который установлен в регистровой паре HL. В общем, шикарный метод для стирания программ. Не совсем то, что мы искали, но хоть что-то. Уж если не попрограммируем, зато хоть постираем от души, благо все делается автоматически.

А ведь если честно, то какое же программирование без хорошего стирания? Вы когда-нибудь хоть строчку написали, чтобы ее потом не стирать? Вот то то же! Всякое правильное программирование всегда начинается со стирания того, что ранее было запрограммировано неправильно.

А уж объединить эти две процедуры в одну программу в машинных кодах - дело нехитрое. Вот и получилась такая миленькая программа, размером всего в 19 байтов, которая хотите верьте, хотите нет, а сэкономит массу времени любому, кто хоть когда-нибудь написал что-то длиннее, чем тридцать строк.

Можете теперь ее встроить в свой собственный "дебют", если хотите, можете держать ее отдельно. Делайте с ней, что хотите, храните в любых адресах, можете встроить в БЕЙСИК-строку, воспользовавшись приемом, которым поделился автор предыдущей статьи.

Мы же, не зная заранее, куда Вы предпочтете ее препроводить, разместили код в буфере принтера, начиная с адреса 23300, а БЕЙСИК-загрузчику выделили строки выше 9990, чтобы под ногами не путался.

Итак, распечатка того, что у нас вышло, помещена в Листинге 1. А распечатка БЕЙСИК-загрузчика приведена в Листинге 2. Самое интересное - то, что самой этой же программой можно стереть и этот БЕЙСИК-загрузчик. Правда, при стирании строки 9996 будет выдано сообщение C: Nonsense in BASIC, но оно уже никакого значения для нас не имеет.

Так что стирайте в собственное удовольствие все, что хотите. Обратите только внимание на нашу статью, посвященную ошибкам в ПЗУ - там в разделе, посвященном функциям пользователя FN () кое-что сказано о том, почему нельзя код для стирания строк оформлять в виде пользовательских функций.

Успешного Вам DELETа!

Листинг 1

```
210000      LD HL,00            ; Номер начальной строки помещается в HL.
                                ; Сейчас там нули, но когда Вы введете
                                ; свое число, оно поступит туда.
CD6E19      CALL 196EH          ; Вызов процедуры ПЗУ для определения
                                ; адреса начала строки.
E5          PUSH HL             ; Запомнили полученный адрес на стеке.
210000      LD HL,00            ; Номер конечной строки помещается в HL.
                                ; Сюда мы тоже введем свое число.
33          INC HL              ; Указание на следующую строку, которая не
                                ; будет удалена.
CD6E19      CALL 196EH          ; Определили адрес конца последней строки.
D1          POP DE              ; Адрес начальной строки сняли со стека в DE.
CDE519      CALL 19E5           ; Удаление строк из заказанного интервала.
C9          RET                 ; Возврат в БЕЙСИК.
```

Листинг 2

```
9991 DATA 33,0,0,205,110,25,229,33,0,0,35,205,110,25,209,205,229,25,201
9992 RESTORE 9991: FOR n=0 TO 18: READ a: POKE 23300+n,a: NEXT n
9993 INPUT "Введите номер начальной СТРОКИ"; a
9994 POKE 23301,a-256*INT(a/256): POKE 23302, INT(a/256)
9995 INPUT "Введите номер конечной строки"; b
9996 IF b<a THEN GO TO 9992
9997 POKE 23308,b-256*INT (b/256): POKE 23309, INT (b/256)
9998 RANDOMIZE USR 23300
```

ВЕКТОРНАЯ ГРАФИКА

Сегодня мы предлагаем Вашему вниманию небольшой отрывок из главы готовящейся сейчас книги "Прикладная графика". Книга является логическим продолжением ранее вышедшего тома "Элементарная графика", а глава, отрывок из которой здесь приведен, посвящена одной из проблем трехмерной векторной графики.

Вам, конечно, неоднократно приходилось сталкиваться с векторной графикой в игровых программах. Практически полностью на ней построена любимая игра тысяч наших читателей "ELITE", та же графика в программах "ACADEMY", "STARION" и в очень необычной, увлекательной программе, требующей тонкого расчета и стратегического мышления - "CENTINELL". Список игр, инкорпорирующих векторную графику, мог бы быть очень и очень обширным и среди них многие относятся к лучшим из лучших.

Вы, уважаемые читатели, обратили, конечно, внимание на то, что векторная графика в этих играх одноцветная, угловатая и художественными достоинствами очевидно не отличается. Так почему же они пользуются таким успехом, с чем он связан?

Да, конечно, векторная графика выглядит на экране победнее, чем многоцветная растровая графика, но у нее есть два огромных преимущества.

Во-первых, это очень быстрая графика. Цикл освежения экрана и перестроения изображения происходит намного быстрее, чем в программах с растровой графикой.

Во-вторых, это вычисляемая графика, то есть не надо хранить в памяти компьютера заранее подготовленные экраны. Все изображения рассчитываются по заданным алгоритмам и практически никогда не повторяются. Благодаря этому Вы можете иметь в таких программах тысячи планетных систем, десятки возможных кораблей противника и нескончаемое разнообразие игровых ситуаций.

Вспомним программу "ELITE". Да, конечно, нужно иметь воображение, чтобы принять угловатую "морковку" на экране Вашего монитора за роскошный корабль "Fer-de-Lance", нашпигованный чудесами науки и техники и отделанный изнутри лучшими породами дерева и самыми дорогими материалами.

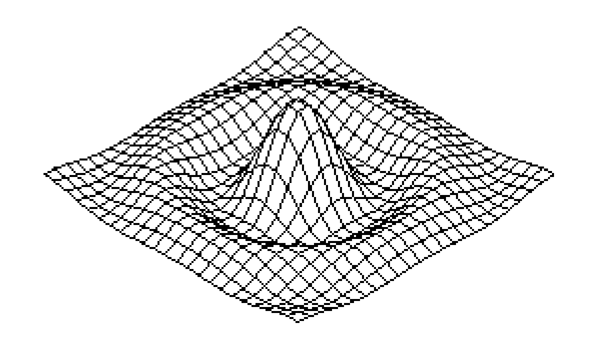


Рис. 1. График функции Z=SIN(R)/R, где
R = SQR (X*X + Y*Y)
Параметры приняты такими, как приведенные в БЕЙСИК-распечатке. Другие параметры дадут иную поверхность.

Но зато когда он вращается вокруг всех собственных осей и при этом летит в пространстве, изменяя свои координаты относительно Вашего корабля, а Вы вместе с ним при этом перемещаетесь и маневрируете относительно планеты, звезды, станции и прочих кораблей и этот клубок пронзают залпы лазеров, в нем летят и находят свою цель ракеты, здесь Вы забываете обо всем - и о "морковке" и о черно-белой графике. Перед Вами реальный, хорошо вооруженный противник - это вызов Вашему мастерству. Динамика игры, острота схватки и неповторяемость ситуаций делают возможным для вас эффект реального присутствия и вам уже не нужно художественное впечатление от богатства красок. Ваш мозг, увлеченный переживаниями, сам домыслит столько, сколько ему надо.

В этой статье мы коснемся только одной маленькой проблемы, которая связана с изображением на экране трехмерной векторной графики. Те из наших читателей, которые захотят скрупулезно изучить вопросы векторной (и не только векторной) графики прочитают книгу, а здесь мы рассмотрим один полезный алгоритм, который может быть принят на вооружение по крайней мере теми, кто использует свой "Спектрум" в практической работе, например при написании курсовых и дипломных проектов.

Сокрытие невидимых линий контура

При работе с трехмерной векторной графикой часто встает одна важная проблема - как изображать невидимые линии трехмерного объекта? Эта задача имеет непосредственное отношение к системам автоматизированного проектирования и наибольшее развитие получила именно в теории этих систем и, надо сказать, для ее решения привлекают довольно сложный аппарат из той области высшей математики, которая называется аналитической геометрией. Для нас с Вами, поскольку мы занимаемся обычной прикладной графикой, эту задачу можно несколько упростить. На данном этапе нас не интересует как изобразить невидимые линии - нам просто нужно их НЕ ИЗОБРАЖАТЬ.

Приемов и методов для достижения этой цели немало и мы рассмотрим один из наиболее простых, поддающийся несложной алгоритмизации.

Посмотрите на рис. 1. На нем изображен некоторый трехмерный ландшафт. Фактически это график функции:

Z = SIN (R)/R, где R = SQR (X*X + Y*Y).

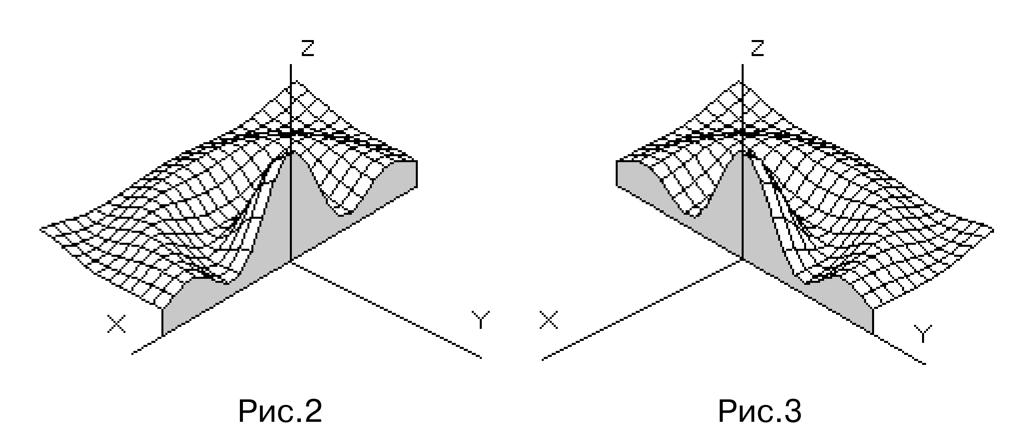


Рис.2 Рис.3

В своих экспериментах вы можете изменить эту функцию и поработать с другими. Важно только, чтобы она имела вид Z=f(x,y), т.е. чтобы была возможность составить однозначный алгоритм для вычисления координаты Z по заданным координатам X и Y. Самое интересное в этом графике - то, что скрытые детали - на самом деле скрыты. Все точки, которые находятся за гребнем или за вершиной - не показаны.

Алгоритм.

Давайте рассмотрим алгоритм, с помощью которого может быть достигнут желаемый эффект. Во-первых, надо отметить, что наша трехмерная поверхность изображается в два приема. На первом проходе изображаются все линии, параллельные оси X (рис. 2), а на втором проходе - линии, параллельные оси Y (рис. 3). Именно благодаря такому порядку изображения кривых и оказывается возможным скрыть невидимые детали.

Линии изображаются с некоторым шагом по X - Xh и по Y - Yh (см. рис. 4). Конечно, чем мельче шаг, тем детальнее будет проработано изображение, но слишком мельчить тоже не надо - существует некоторый оптимум, который можно установить методом проб и ошибок. Во всяком случае, параметры

Xh=(Xmax-Xmin)/20

и

Yh=(Ymax-Ymin)/20

выглядят достаточно удачными.

Если какая-то часть вычерчиваемой в текущий момент линии оказывается за ранее проведенной кривой, то она не изображается и есть достаточно простой прием, который позволяет в программе принять такое решение.

Если мы выстраиваем изображение в виде семейства кривых, начиная от ближайшей к наблюдателю и удаляясь от него назад (т.е. идем от т.т. X2 и Y2 к т.т. X1 и Y1, как показано на рис. 4), то фактически те точки текущей линии, которые оказывается на экране ниже, чем точки ранее изображенных линий и являются невидимыми и должны быть скрыты.

Чтобы решить этот вопрос программно, мы создаем в оперативной памяти буфер размером 256 байтов, а дальше действуем следующим образом. Поскольку экран "Спектрума" имеет в ширину 256 пикселов, то мы будем считать, что он образован как бы из 256-ти узких однопиксельных вертикальных столбцов. Каждому столбцу отведем по одной ячейке памяти в нашем буфере и теперь всякий раз, когда будем печатать на экране точку, будем смотреть, что же содержится в буфере для данного столбца. Если то значение, которое есть там - меньше, чем вертикальная координата экрана, в которой мы будем печатать точку, то новая координата запоминается в данном буфере и точка печатается. Если же хранящееся там значение больше, чем текущая вертикальная координата позиции печати, то точка должна быть скрыта и не печатается, а значение в буфере не изменяется.

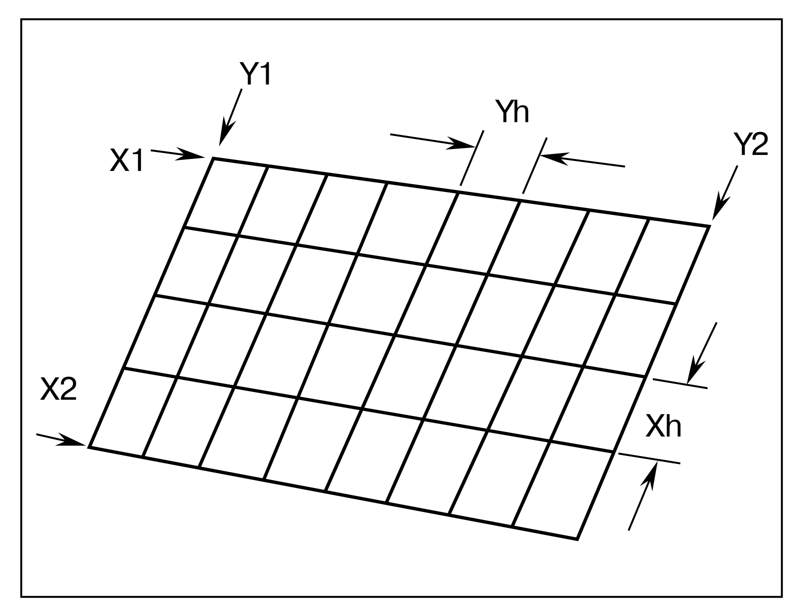


Рис. 4

Итак, алгоритм имеет следующий вид:

1. Задаем максимальные значения координат X2 и Y2.
2. Задаем минимальные значения координат X1 и Y1.
3. Определяем шаг по осям X и Y - Xh,Yh.

Xh =(Xmax-Xmin)/23
Yh=(Ymax-Ymin)/23

Мы специально делили здесь на "некруглое" число 23, а не на 20. Это помогает избежать прохождения через нулевую точку в том случае, если максимальные и минимальные параметры заданы симметрично относительно нуля. Все-таки неуютно себя чувствуешь, когда программа должна посчитать SIN(R)/R, когда R равно нулю. Хотя в принципе и этот случай можно было бы предусмотреть. Для тех, кто еще пока не изучал высшую математику, подскажем, что SIN(R)/R приближается к единице, когда R стремится к нулю.

4. Определяем масштаб по осям X и Y.

Для системы координат, показанной на Рис. 1...3 масштаб по X и Y - одинаков. Он зависит от ширины экрана и от угла между осями X, Y и осью Z. Для того, чтобы при любых

допустимых значениях X и Y точка умещалась бы на экране, нам необходимо избрать масштаб:

(X2-X1) + (Y2-Y1)*255/SQR(3)*2

5. Задаем масштаб для оси Z (его можно менять).

6. Создаем буферный массив из 256 элементов, обнуляем их.

7. Начинаем строить семейство кривых, "параллельных" оси X. Организуем цикл по Y от Y2 до Y1 с шагом Yh.

8. Внутри этого цикла строим кривую "параллельную" оси X. Организуем цикл от X2 до X1 с шагом по Xh.

9. Внутри этого цикла для текущих значений X и Y определяем Z по заданной формуле функции.

10. Полученный результат для Z умножаем на масштаб, получаем координату Z для графика.

11. Выполняем преобразование систем координат. По трехмерным координатам X, Y, Z находим значения X и Y для плоскости экрана.

Формулы для этого преобразования будут зависеть от того, какую проекцию трехмерной системы координат на плоскость Вы изберете. Другими словами, они зависят от того, под какими углами Вы смотрите на трехмерный объект. Для случая, показанного на Рис. 1.. Рис. 3 подойдут формулы:

X = SQR(3)* (Y-X)/2 + 127

Y = Z - (Y+X)/2 + 87

12. Мы готовы поставить на экране точку в координатах x, y. Но сначала проверим, что есть в буфере для данной координаты x. Если там значение меньше, чем y, то точку x, y на экране ставим и значение y обновляем в буфере, а если оно больше, то точка - невидима, мы ее не ставим и значение в буфере не обновляем.

13. Вычислив экранные координаты точки x, y, мы готовы соединить ее линией с предыдущей точкой x', y' (если наша точка не первая). Хорошо бы для этого воспользоваться командой БЕЙСИКа DRAW или процедурой изображения отрезков в машинных кодах, но делать этого, к сожалению, нельзя. Причина в том, что этот отрезок (или его часть) может быть невидимым. Значит, надо строить его по точкам и для каждой точки проверять по буферу видима она или нет. Поэтому опять же надо организовать цикл для изображения отрезка по точкам.

14. Теперь надо определиться с параметром этого цикла. Он может изменяться по горизонтали (по x), а может и по вертикали (по y). Надо понять, что больше - приращение dx (равное x-x') или dy (равное y-y'). То, которое больше, и следует принять в качестве параметра цикла.

15. Определившись с параметром, организуем цикл и внутри него вычисляем координаты текущих точек, проверяем для них y сравнением с буфером и, если точка видима, печатаем ее и обновляем буфер, а если нет, то не печатаем и не обновляем буфер.

Здесь есть маленькая хитрость, которая несколько усложняет жизнь программисту. Дело в том, что эти соединительные отрезки можно проводить слева-направо, а можно и справа налево. В принципе это все равно, но есть один нюанс. Допустим мы будем их проводить слева направо. Все будет в порядке, пока нам не придется провести круто падающий отрезок. На один шаг по x для него происходит несколько шагов по y. И бывает так, что одной координате x соответствуют несколько точек y. Если бы мы рисовали этот отрезок снизу вверх, все было бы в порядке, а при движении сверху вниз (слева-направо) ранее напечатанная точка может "блокировать" печать следующих, забив в буфере свое координату. Поэтому круто падающие отрезки программа должна строить наоборот - справа-налево. Тогда отрезок становится как бы не "падающим", а "восходящим".

16. Соединив две точки на экране, переходим к очередной точке, отстоящей на Xh, и возвращаемся на шаг 8.

17. Построив кривую, "параллельную" оси X, переходим к следующей, отстоящей от нее на шаг Yh. Возвращаемся на шаг 7.

18. Когда все семейство кривых, "параллельных" оси X построено, половина дела

сделана. Теперь надо построить семейство кривых, параллельных оси Y.

19. Для этого сначала переинициализируем буфер, обнулив все его значения, а затем повторим все то же, что мы делали для семейства кривых, идущих вдоль оси X (шаги 7 - 18). Правда, при этом вместо шагов по X будем делать шаги по Y и наоборот.

```
10 LET xmax = 10: LET ymax = 10
20 LET xmin =-10: LET ymin = -10
30 LET xh = -(xmax-xmin)/23: LET yh = -(ymax-ymin)/23
40 LET scale = 255/SQR(3)*2/((xmax-xmin) * (ymax-ymin))
50 LET zscale = 40
60 DIM c(256): FOR i = 1 TO 256: LET c(i)=0: NEXT i
70 FOR y=ymax TO ymin STEP yh: LET y1 = y*scale
80 FOR x=xmax TO xmin STEP xh: LET x1=x*scale
90 GO SUB 5000
100 IF x=xmax THEN GO SUB 8100: NEXT x
110 LET dy=yt-yold: LET dx=xt-xold
120 IF ABS(dy) >= ABS(dx) THEN GO SUB 6000: GO TO 140
130 GO SUB 7000
140 NEXT x
150 NEXT y
155 FOR i=1 TO 256: LET c(i)=0: NEXT i
160 FOR x=xmax TO xmin STEP xh: LET x1 = x*scale
170 FOR y=ymax TO ymin STEP yh: LET y1 = y*scale
180 GO SUB 5000
190 IF y=ymax THEN GO SUB 8100: NEXT y
200 LET dy=yt-yold: LET dx=xt-xold
210 IF ABS(dy) >= ABS(dx) THEN GO SUB 6000: GO TO 230
220 GO SUB 7000
230 NEXT y
240 NEXT x
250 STOP
4997 REM
4998 REM*********************************************
4999 REM
5000 LET r= SQR(x*x + y*y) : LET z = SIN(r)/r
5010 LET z = z*zscale
5020 LET xt=127+SQR(3)*(y1-x1)/2
5030 LET yt=87+z-(y1+x1)/2
5040 RETURN
5997 REM
5998 REM*********************************************
5999 REM
6000 IF dy<0 THEN GO TO 6100
6010 FOR i=0 TO dy
6020 GO SUB 6500
6030 NEXT i
6040 GO SUB 8100: RETURN
6100 FOR i=dy TO 0
6110 GO SUB 6500
6120 NEXT i
6130 GO SUB 8200: RETURN
6497 REM
6498 REM************************************************
6499 REM
6500 LET xt = xold+dx/dy*i
6510 LET yt=yold + i
6520 GO SUB 8000: RETURN
6997 REM
6998 REM************************************************
6999 REM
7000 IF dx<0 THEN GO TO 7100
7010 FOR i=0 TO dx
7020 GO SUB 7500
```

```
7030 NEXT i
7040 GO SUB 8100: RETURN
7100 FOR i=dx TO 0
7110 GO SUB 7500
7120 NEXT i
7130 GO SUB 8200: RETURN
7497 REM
7498 REM ************************************************
7499 REM
7500 LET xt=xold+i
7510 LET yt = yold+dy/dx*i
7520 GO SUB 8000: RETURN
7997 REM
7998 REM************************************************
7999 REM
8000 IF yt > c(xt+1) THEN LET c(xt+1)=yt: PLOT xt,yt
8010 RETURN
8097 REM
8098 REM************************************************
8099 REM
8100 LET xold=xt: LET yold=yt: RETURN
8197 REM
8198 REM************************************************
8199 REM
8200 LET xold=xold+dx: LET yold=yold+dy: RETURN
```

СПИСОК ПРОГРАММНЫХ ПЕРЕМЕННЫХ

xmax - максимально-допустимое значение координаты X (задаетсяпользователем).

ymax - то же для координаты Y.

xmin - минимально-допустимое значение координаты X (задается пользователем).

ymin - то же для координаты Y.

xh,yh - шаг между узлами сетки.

scale - масштаб по координатам X и Y (зависит от углов в пространстве, под которыми наблюдатель смотрит на трехмерный объект). Рассчитывается исходя из соображений оптимального использования плоскости экрана.

zscale - масштаб по оси Z (задается пользователем "по вкусу", но так, чтобы изображение по вертикали не вышло за пределы экрана). Возможно и автоматическое определение zscale в программе, во для простоты это не было сделано.

c(256)- буферный массив на 256 элементов.

x1, y1 - отмасштабированные значения трехмерных координат X и Y.

r - математический комплекс, нужный для вычисления Z.

x,y - текущие трехмерные координаты X и Y в узлах сетки.

xt,yt - текущие экранные координаты (двумерные).

xold,yold - экранные (двумерные) координаты предыдущего узла.

dx,dy - приращения экранных координат на очередном шаге (расстояние на экране между узлами).

* * *

Вот практически и весь алгоритм. Его описание выглядит страшнее, чем текст программы на БЕЙСИКе (см. распечатку), и это не случайно - ведь БЕЙСИК гораздо лучше подходит для описания алгоритмов и программистских идей, чем нормальный человеческий язык.

Конечно, скорость работы этой программы на БЕЙСИКе оставляет желать лучшего, но для иллюстрации самой концепции он неплох.

Мы попробовали - у нас получилось время работы программы что-то порядка пятнадцати минут, но тем не менее не пожалейте этого времени, поэкспериментируйте с

графиком.

Вы можете менять максимальные и минимальные значения X и Y. Вы можете менять масштаб по Z. В наших экспериментах мы получали кроме представленной на рис. 1 "шляпы" еще и "верблюда", "русалку", "замок в горах", "пепельницу", "ковер-самолет" и пр. и пр.

Более того, Вы можете менять и саму функцию, исследуя другие поверхности. Если Вы в приведенном нами примере замените вторую степень при X и Y на четвертую, то почувствуете, как холодок бежит по спине, когда компьютер изобразит очень натуральную свежую могилку с не менее натуральным каменным валуном в изголовье. Хочется пожелать Вам найти что-либо менее мрачное, особенно если вы работаете с компьютером по ночам.

Те же, кто предпочтут использовать для программирования машинный код, получат прекрасные результаты, но надо учесть, что программа выполняет большой объем чисто математических вычислений (это вообще характерно для трехмерной векторной графики). Здесь и масштабирование (чтобы график аккуратно занимал плоскость экрана) и пересчет из одной системы координат в другую (из трехмерной системы координат в двумерную систему координат плоскости экрана) и, конечно же, расчет самой функции Z=f(X,Y).

Процессор Z-80 не может оперировать с действительными числами, не может выполнять математических расчетов и здесь используют программирование в кодах калькулятора.

Пример программы в машинном коде мы здесь не даем, оставив для тех, кто интересуется, возможность ознакомиться с ним в книге. Подробный комментарий всех входящих процедур явится хорошим пособием для тех, кому необходимо программирование в кодах калькулятора.

Маленькие хитрости

В этом разделе мы коснемся некоторых путей повышения быстродействия часто встречающихся операций.

Так, например, в предыдущей статъе мы с Вами затронули маленький вопрос об очистке 256-байтного буфера. Давайте посмотрим, как бы мы делали эту операцию, если бы программировали в машинном коде.

Мы бы загрузили в регистровую пару HL адрес NN, с которого начинается наш буфер. Затем в регистре B организовали бы счетчик на 256 байтов (FFH), обнулили бы аккумулятор командой XOR и затем в цикле поместили бы содержимое аккумулятора в ячейки буфера, на которые указывает HL. При этом на каждом шаге увеличивали бы HL на единицу.

Мы специально пишем столь подробно об этих элементарных вещах, потому что рассчитываем, что нас могут читать и те, кто только подумывает об освоении машинного кода.

```
      LD HL,NN         (10)
      LD B, FFH        (7)
      XOR A            (4)
LOOP  LD (HL),A        (7)*256
      INC HL           (6)*256
      DJNZ LOOP        (8)*256
```

Давайте посмотрим сколько времени займет это мероприятие. В скобках проставлено время, необходимое для выполнения каждой из приведенных команд процессора. Это время измеряется в тактах работы процессора, а пересчет в секунды возможен, если знать частоту задающего генератора в вашем компьютере. Но нам достаточно и тактов, чтобы сравнить между собой различные приемы.

Итак, приведенный выше пример займет: 21 + 21*256 = 5397 тактов, т.е. в среднем 21,08 такта на очистку одного байта в буфере.

Можно ли быстрее? Скорее проще, чем быстрее. Те, кто знают машинный код, осведомлены о наличии команды LDIR, которая служит для автоматической очень быстрой переброски блоков данных из одной области памяти в другую. Эта команда перебрасывает блок байтов, длина которого установлена в регистровой паре BC из области, начинающейся с адреса, установленного в регистровой паре HL в область, на начало которой указывает содержимое DE. Остается открытым вопрос, а как можно использовать LDIR для очистки, ведь если эта команда может быстро перебросить блок нулевых байтов, то ведь такой блок надо сначала создать - в общем придем к тому, от чего ушли (мы не можем в общем случае рассчитывать на то, что где-то в компьютере есть пространства с нулевым содержимым ячеек, откуда можно черпать "пустые" массивы).

Оказывается, использовать LDIR все же можно, хоть и делается это несколько необычно. Рассмотрим пример:

```
LD HL, NN        (10)
LD DE,NN+1       (10)
LD BC, 00FFH     (10)
LD (HL),B        (7)
LDIR             (21)*256
```

То, что здесь происходит, может показаться чепухой. В HL установили адрес начала нашего буфера, в DE - адрес второго байта буфера, в BC - счетчик на 255. Командой LD (HL),B очистили первый байт буфера, а потом зачем-то передвинули все содержимое буфера на один байт вверх. Ну получим в итоге, что и второй байт станет нулевым, а дальше-то что?

Вся хитрость состоит в том, что во время работы команды LDIR содержимое HL, DE и BC не остаются неизменными. После переброски каждого очередного байта HL и DE

увеличиваются на единицу, а BC на единицу уменьшается. Благодаря этому после переброски первого байта во второй, DE уже укажет на третий, а HL "подхватит" нулевой байт из второй ячейки и так далее. В общем, весь буфер будет вычищен.

Итак, при работе с командой LDIR мы затратим: 37+21*256= 5413 тактов - это хоть и не быстрее, но, главное, изящнее. Получается в среднем 21.14 такта на байт.

Те, кто любят "маленькие хитрости", наверное, ждут, что мы предложим что-либо еще более скоростное, чем LDIR. Возможно, что большинство профессионалов сказали бы, что это невозможно, ничего быстрее LDIR для манипуляции с большими блоками памяти не бывает.

Но нет, оказывается, бывает, причем не просто быстрее, а быстрее в несколько раз. Есть одно гениальное решение, которое мы не постесняемся так назвать, поскольку не мы его придумали, а "выудили" его в результате анализа машинного кода программы "STARION" фирмы "Melbourne House". На поиски натолкнул тот факт, что программа в части межзвездных сражений не уступает "ELITE", а вот графика более динамичная, гладкая и плавная. Там это решение применяется не для очистки маленьких таблиц, а для освежения всей экранной памяти, а ее размер велик и там каждая тысячная доля секунды на счету.

Рассмотрим используемый алгоритм на нашем примере с очисткой 256-байтного буфера.

```
      LD (MM),SP      (20)
      LD HL,0000      (10)
      LD SP,KK        (10)
      LD B,80H        (7)
LOOP  PUSH HL         (11)*128
      DJNZ LOOP       (8)*128
      LD SP,(MM)      (20)
```

Вся хитрость состоит в использовании стека. Исходное положение вершины стека надо запомнить в каком угодно адресе MM затем приготавливаем регистровую пару HL - обнуляем ее. Новый стек организуем в нашем буфере. Но надо помнить, что стек в памяти компьютера "растет" сверху вниз. Поэтому, чтобы очистить весь буфер, мы должны начать не с его начала, а с конца. LD SP, KK - прогружает в качестве вершины стека конец нашего буфера. Счетчик байтов создаем в регистре B, как обычно, но поскольку здесь байты будут перебрасываться не по одному, а парами, то счетчик надо настраивать не на 256, а на 128 - (LD B,80H) перемещений.

Все подготовительные операции сделаны. Теперь в цикле 128 раз помещаем содержимое HL на вершину стека, то есть обнуляем буфер. Закончив операцию, надо не забыть восстановить старое значение указателя стека, временно сохраненное в ячейке MM.

Всего мы затратили:

67+19*128 = 2499 тактов, т.е. примерно по 9.76 такта на байт!

Встает старый вопрос: "А нельзя ли еще быстрее?"

Пожалуйста, отвечаем мы. Идею Вы уловите сами очень легко и поймете, в каком направления надо идти, из следующего примера:

```
      LD (NN),SP      (20)
      LD HL,0000      (10)
      LD SP,KK        (10)
      LD B,20H        (7)
LOOP  PUSH HL         (11)*32
      PUSH HL         (11)*32
      PUSH HL         (11)*32
      PUSH HL         (11)*32
      DJNZ LOOP       (8)*32
      LD SP,(MM)      (20)
```

Всего имеем: 67+52*32 = 1731 - по 6,76 такта на байт.

Подумать только, с чего мы начинали! А ведь это еще не предел. При приличном

количестве команд PUSH HL, как в STARION'е, можно работать быстрее, чем с LDIR в четыре раза! И, конечно, такая скорость не нужна для освежения буфера - на самом деле это нужно для работы с экраном.

Прощаясь, признаемся, что там же мы "выудили" прием не только очистки, но и перестроения экрана, работающий в два раза быстрее, чем LDIR. В основу положена та же идея манипуляций со стеком, хотя сделано это намного сложнее. При случае мы вернемся к этому вопросу, хотя многие наверное уже и сами сообразят, как это должно быть. У нас ведь народ тертый, ему только намекни...

ОШИБКИ ПЗУ

Обзор по материалам зарубежной печати

В этой статье рассмотрены ошибки и неточности, имеющиеся в ПЗУ "СПЕКТРУМА". Может быть не все они, строго говоря, и являются ошибками, а просто особенностями компьютера, но о них надо знать и уметь обходить в тех случаях, когда они могут помешать нормальной работе.

1. Ограничение по использованию регистровой пары IY.

Эта ошибка (неточность) связана с тем, как в "Спектруме" обрабатываются маскируемые прерывания. Вы знаете, что обычно компьютер 50 раз в секунду приостанавливает исполнение своей текущей программы и обращается по адресу 0038H = 56 DEC для запуска процедуры обработки маскируемого прерывания. Эта процедура увеличивает на единицу показания системных часов компьютера (трехбайтную системную переменную FRAMES) и вызывает подпрограмму сканирования клавиатуры в поисках нажатой клавиши. Благодаря этому и возможен диалог между Вами и компьютером.

Ошибка связана с тем, что увеличение на единицу системных часов производится некорректно. Младшие два байта увеличиваются командой INC HL и с ними все в порядке, но когда они переполняются и надо увеличить старший байт, расположенный по адресу 5C7AH (33674), для этого используется команда процессора

```
INC (IY+40)
```

Программист, который писал эту процедуру, предполагал, что во время ее работы в регистровой паре IY должно быть значение 5C3AH (23610) - указание на системную переменную ERR-NR. в принципе так оно и должно бы быть.

Когда вы вызываете подпрограмму, написанную в машинном коде с помощью RANDOMIZE USR nn, процедура калькулятора, занимающаяся обработкой функции USR nn (она расположена по адресу 34B3H) автоматически сохраняет на стеке адрес 2D2EH. Поэтому, когда Вы вернетесь по RET из своей подпрограммы, то попадете сначала в адрес 2D2BH (процедура STACK_BC). Здесь восстанавливается нормальное содержимое пары IY = 5C3AH.

Таким образом, у Вас все было бы в полном порядке, если бы не одно "но". Ведь во время работы самой Вашей процедуры тоже может пройти системное прерывание. И если Вы изменили содержимое IY, то вместо приращения системных часов получите что угодно.

Ошибка ПЗУ состоит в том, что в процедуре обработки маскируемого прерывания следовало бы выставлять в IY значение 5C3A, а только потом наращивать старший байт системных часов, а перед выходом восстанавливать в IY Ваше значение.

Достаточно обидно иметь в своем распоряжении регистровую пару IY и не иметь возможности ею воспользоваться. И пользоваться ею можно, если при этом соблюдать некоторые меры предосторожности.

1. Если Вы намерены использовать регистровую пару IY, то возьмите себе за правило прежде, чем что-либо изменять в этой паре, отключать системные прерывания командой процессора DI - DISABLE INTERRUPTS, а перед выходом из процедуры восстанавливать их командой EI - ENABLE INTERRUPTS.

2. Может быть, Вам захочется даже создать свою собственную процедуру для обработки прерываний.

2. Особенности регистровой пары H'L' (альтернативной).

Эта ошибка коснется тех, кто программирует в машинных кодах, а еще точнее - тех, кто в своих программах объединяет БЕЙСИК и машинный код, а так поступают очень многие.

Они пишут логику программы на БЕЙСИКе, а операции, требующие большой скорости, пишут в машинном коде. Вероятность споткнуться об эту ошибку практически стопроцентная. Рано или поздно она испортят Вам кровь, хотя избежать ее очень просто.

В двух словах, суть состоит в том, что если Ваша процедура, написанная в маш. коде изменит содержимое регистровой пары H'L' (альтернативной), то в БЕЙСИК Вы уже не вернетесь и программа сбросится или зависнет.

Почему так происходит? Чтобы ответить на этот вопрос, надо знать, как происходит вызов пользовательских машиннокодовых процедур. Вы конечно знаете, что они вызываются командой RANDOMIZE USR nn, где nn - адрес ее старта. В принципе вместо RANDOMIZE USR может использоваться RESTORE USR, PRINT USR и т.п., сейчас нам это все равно. Суть состоит в том, что когда интерпретатор БЕЙСИКа встретит USR nn, то он должен рассчитать чему равно nn и для этого вызывает встроенный калькулятор. Это делает процедура ПЗУ, которая занимается расчетом выражений. По адресу 2756H она содержит вызов калькулятора, а по адресу 2758H - выход из калькулятора.

При вводе в калькулятор содержимое вершины машинного стека (а это есть адрес возврата из калькулятора) запоминается в альтернативной регистровой паре H'L'. Это делает процедура, находящаяся по адресу 3362H.

Затем процедура калькулятора, занимающаяся обслуживанием функции USR nn (34B3H) помещает на вершину стека адрес 2D2Bh и адрес "nn".

По команде RET со стека снимается адрес "nn" - выполняется переход в Вашу машиннокодовую процедуру.

Когда она отработает, по ее команде RET со стека снимется адрес 2D2BH и выполнится переход по этому адресу в ПЗУ. Мы писали, рассматривая предыдущую ошибку о том, что расположенная там процедура восстанавливает "стандартное" значение в регистровой паре IY, но она ничего не делает, чтобы восстановить содержимое альтернативной пары H'L', а ведь она тоже могла быть нарушена при исполнении пользовательской процедуры.

Таким образом, исполнение пользовательской процедуры в машинных кодах начинается с вызова системного калькулятора для расчета адреса старта, а кончается выходом из него. Выход из калькулятора производится помещением на стек того, что хранилось в H'L' и переходом по этому адресу. Это делает процедура, обслуживающая команду калькулятора end-calc, расположенная по адресу 369BH. И содержаться в этот момент в H'L' может только адрес 2758H.

Итак, если Ваша процедура нарушает содержимое регистровой пары H'L' (альтернативная) и Вы рассчитываете вернуться в БЕЙСИК, то потрудитесь перед выходом из процедуры восстановить в этой паре значение 2758H, иначе ничего у Вас не получится.

3. Особенности пользовательской функции FN.

Недоразумение, связанное с пользовательскими функциями может коснуться и тех, кто программирует на БЕЙСИКе и тех, кто программирует в машинном коде, но скорее всего с ним столкнутся опять те же энтузиасты, которые используют и БЕЙСИК и машинный код одновременно, причем пользуются для передачи параметров из БЕЙСИКа в маш. код пользовательскими функциями. Это весьма удобный и прогрессивный метод и мы подробно осветили его в первом томе, посвященном графике - "Элементарная графика".

Суть этой ошибки состоит в том, что если во время исполнения пользовательской функции произойдет какое-либо перемещение программной области БЕЙСИКа вверх или вниз, то будет выдано сообщение C; Nonsense in BASIC.

Конечно, не так-то просто выполнить перемещения БЕЙСИК-области и "нарваться" на такую ошибку, но это вполне возможно, если Вы используете USR внутри вашей функции.

На практике это может означать вот что. Если Вы, например, зададите пользовательскую функцию FN D(m,n) = USR NN, предназначенную для удаления из БЕЙСИК-программы строк m и n, то она у вас работать не сможет.

Причина появления ошибки - в статическом характере системной переменной CH ADD (23645=5C5DH), которая при работе интерпретатора содержит адрес следующего

интерпретируемого символа. На время исполнения пользовательской функции она "сохраняет" свое состояние на стеке и, если во время работы функции все адреса, имеющие отношения к БЕЙСИКУ "поплывут", то после окончания ее работы интерпретатор не сможет продолжить исполнение БЕЙСИК-программы.

4. Ошибка деления.

Эта ошибка встречается настолько часто, что о ней даже и не говорят, как об ошибке деления, а считают ее ошибкой округления. Однажды в разделе "ФОРУМ" мы отвечали на письмо читателей, связанное с этой ошибкой (см. "ZX-РЕВЮ-91, с. 222).

Кратко проиллюстрировать ее можно на примере:

```
IF 1/2 <> 0.5 THEN PRINT "Ku-Ku"
```

Попробовав, Вы сами убедитесь, что 1/2 не равна 0.5.

Конечно, в инженерных расчетах следует вводить понятие точности вычислений и сравнивать действительные числа между собой только в пределах установленной точности, но все-таки где-то что-то в ПЗУ не совсем то, раз компьютер выдает такие непонятные результаты.

Ошибка содержится в процедуре калькулятора, выполняющей деление и связана с неправильным округлением результата. Не вдаваясь в подробности как происходит деление двух действительных чисел, скажем только, что процедура работает с 32-мя битами, что и определяет точность числа. Но "за кадром" она учитывает еще 33-ий и 34-ый биты для правильного округления. Так вот, именно при добавлении 34-го бита и происходит ошибка - он не учитывается из-за неправильного перехода в адресе 3200H. Там записана команда JR Z, 31E2, а должно быть JR Z, 31DB.

Наилучший путь обхода - установить требуемую точность для своих расчетов, например:

```
LET eps = 0.0000001
```

и затем вместо

```
IF X = Y
```

употреблять:

```
IF (X-Y) < eps
```

5. Ошибка "-65536".

Это довольно известная ошибка. Она легко вызывается оператором:

```
PRINT INT(-65536)
```

который дает -1.

Причина скрыта в процедурах калькулятора, которые работают с пятибайтной формой представления чисел. Одни из них считают, что целые числа могут быть только в диапазоне от -65535 до +65535.

Некоторые, например процедура, выполняющая сложение, полагают, что и -65536 - тоже допустимое целое число, в частности, возможно, что сложив -65000 и -536, можно получить число -65536, которое внутри компьютера будет представлено не как действительное, а как целое число.

Процедура же INT - одна из тех, которые " разоблачают" эту двусмысленность.

В известной книге Я. Логана и Ф. О'Хары "The Complete SPECTRUM ROM Disassembly" подробному разбору этой ошибки посвящена целая глава приложения. Объяснения достаточно мудреные для непрофессионалов и мы отошлем тех, кому эти знания существенно важны, к этой книге.

6. Ошибка оператора PLOT.

Может быть, это и не ошибка в том смысле, что к фатальным последствиям она не приведет. Дело в том, что если в операторе PLOT X,Y Вы зададите координаты X и Y отрицательными числами, то все равно получите точку на экране.

На самом деле получается так, что команда PLOT печатает точку не в координатах X и Y, а в координатах ABS(X),ABS(Y). Причина - в том, что процедура ПЗУ PLOT (22DCH)

вызывает процедуру, перемещающую целые числа из вершины стека калькулятора в регистровую пару BC (процедуру STK_TO_BC - 2307H). Но после того, как STK_TO_BC отработает, не учитывается, что она еще поместила в регистровую пару DE знаки тех чисел, которые пошли в BC, и при возвращении в процедуру PLOT надо было бы проверить DE на знак, что не было сделано.

7. Ошибка первого экрана в компьютерах 128K.

Мы уже писали об этой ошибке в предыдущем выпуске "ZX-РЕВЮ" на стр. 156-159. Напомним, что 128-килобайтные машины имеют две экранных области памяти. Нулевой экран расположен в обычных адресах - 4000H, а дополнительный первый экран - в адресе 7C000H. Их можно переключать. Команда POKE 23388,24 включит первый экран, а команда POKE 23388,16 включит нулевой экран.

Ошибка происходит в тех случаях, когда пользователь работает с верхними областями памяти этой машины, как с электронным RAM-диском и сохраняет в ней данные в виде многочисленных файлов. Файлы "располагаются" в памяти снизу вверх и в это же время сверху вниз в памяти растет каталог этих файлов. В компьютере ничего не сделано для того, чтобы каталог или сами файлы не "перехлестнулись" с первым (дополнительным) экраном.

Чтобы этого не произошло, не храните более 216 файлов, как бы малы они не были, чтобы экран не нарушился каталогом. Вам также нельзя иметь в сумме более 64K данных в этих файлах, иначе экран-1 будет испорчен самими файлами.

(Окончание в следующем выпуске).

Blinky's



Сегодня мы представляем Вашему вниманию экспертную проработку программы BLINKY'S. Ее сделал наш читатель из г. Днепропетровска - Садошенко Денис. Вы сами увидите, что исполнена она в необычном жанре - назовем его "компьютерной новеллой". Денис пишет, что его друзьям и близким идея очень понравилась. Понравилась она и нам и мы надеемся, что понравится и Вам, уважаемые читатели.

Программа относится к жанру аркадных адвентюр (ARCADE/ADVENTURE). Должны сказать, что с экспертизой этого жанра у нас всегда были проблемы. И дело вовсе не в том, что разобраться с этими играми и пройти их от начала до конца не всегда просто. У нас так много поклонников этого жанра, что всегда есть кто-то, кто разобрался с той или иной игрой. Трудность состоит в том, как представить результаты своего исследования. Ведь если просто расписать подробный порядок прохождения игры, то и читать и играть будет неинтересно.

В свое время мы пробовали представить описание аркадной адвентюры в виде многоуровневой системы подсказок (см. "PIJAMARAMA" в ZX-РЕВЮ-91, стр. 255) и полагали этот вариант удачным, но сейчас должны признать, что идея Д. Садошенко выглядит привлекательнее.

Нам кажется, что ему удалось найти тот баланс, который позволил сочетать и художественную привлекательность и фактическое содержание. Надеемся, что и наши читатели это оценят.

Садошенко Д.
г. Днепропетровск.

Здравствуйте, дорогие читатели!

Это еще одна запись в моем дневнике. Как Вы знаете, меня зовут BLINKY'S и я, если честно, не совсем живой человек, я - привидение. Сейчас вот сижу у любимого камина, в котором догорают сосновые брусочки и вспоминаю свое прошлое.

Давным-давно жил я в вашей жизни, где есть зеленая трава, деревья, прекрасный чистый воздух и прочие блага. Здесь, в потустороннем мире такого не увидишь. Служил я у своего хозяина - короля придворным шутом. Мы с его слугами жили в старинном замке, который достался королю от деда. При дворе жилось легко и привольно. У меня была постоянная работа и постоянная еда. А большего и не надо. Трудности начинались, когда в замок приезжал очередной гость, ведь наш король был очень общительным человеком. Приходилось работать в поте лица, веселя гостей, за что нередко получал в качестве подачки еду и мелкие монетки. Так бы и продолжалась моя беззаботная жизнь, но вот беда - не любила меня старая королева, вторая жена короля. Может, и не напрасно не любила, иногда я вместо того, чтобы веселить гостей, развлекался сам, делая разные гадости королю и королеве. Например, на кухне я перепутал все специи и рассовал их по баночкам с разными надписями, так что в баночке с этикеткой "соль" можно было увидеть перец, а с этикеткой "сахар" - корицу. Это было так здорово!

За эту в принципе безобидную выходку меня выпороли. Вдобавок я еще и лишился работы на неделю, так как при виде меня, покрытого синяками и ссадинами, все плакали, а не смеялись. Отношение короля ко мне ухудшалось с каждым днем. А я день ото дня все больше наглел. Дошел до того, что стал изображать его во всех подробностях - с короткой левой ногой, плешью и тупым выражением лица. Королю донесли, и меня бросили в тюрьму.

На следующий день состоялся местный суд. Прокурором была королева и этим все уже сказано. Меня приговорили к смерти, и в тот же день приговор привели в исполнение…

Вот так я оказался среди мертвых. И нахожусь здесь уже более 4 лет. Каждый год в день моей смерти здесь отмечается небольшой праздник. Прилетает много моих друзей, которых я знал давно, и которые умерли до меня. И каждый год ровно на три часа (с 12 до 3 ночи) меня отпускают к живым.

Обычно я прилетаю в родной дворец и шляюсь по нему. Меня никто не видит, а я вижу всех. Но по ночам, когда все спят, из темных углов вылезают пауки, мыши, улитки. Вылетают вампиры и осы. Проклятый замок стареет с каждым днем - с потолка капает вода, сыпется песок и падают тяжелые плиты. Возле дворца есть ров с водой. Там живут зубастые пираньи, которые никогда не спят. Даже ночью они плавают, отражая тусклыми глазами рассеянный свет далекой луны.

Очень интересно и опасно ночью во дворце. Целый год я жду этой ночи, и каждый раз она себя оправдывает. Вот и та история, которую я хочу вам рассказать, была очень смешно и хитро задумана.

Я решил разбудить и сильно напугать старого короля. Ну что еще может сделать привидение!? Коварно выбрал время - 3 часа утра. В этот час он наиболее крепко спит, а у меня как раз кончается "отпуск".

Сделать это вроде бы и нетрудно, но пришлось попотеть. Каждое прикосновение к живому существу, будь то человек или мерзкий паук, отбирает у меня немного магической энергии. Если энергии не хватит, или я задержусь в замке до рассвета, тогда одним привидением на земле станет меньше. Меня просто не будет ни среди живых, ни среди мертвых. Вот так.

Но я был полон оптимизма и от всей души взялся за дело. На эту ночь, в качестве подарка, мне дали 4 дополнительных жизни. Если я использую их все до 3 утра, то никогда больше не вернусь к своим мертвым друзьям. Однако, во время прошлых посещений замка я научился избегать неприятных контактов, ведущих к повторной смерти. Даже привидение можно убить. Поэтому я был очень осторожен в этот раз. Мне совсем не хотелось раствориться в ночной воздухе из-за малейшей небрежности.

Почти всегда после перемещения я оказываюсь в подвале, в самом темном месте. А король спит наверху в своей любимой башне. Приходится проходить длинный и опасный путь. Во дворце летать я не могу, т.к. там очень тесно. Могу только ходить и прыгать.

Замок кишит тайными ходами - например, есть подводный ход, ведущий в замок снаружи. Нечисть даже построила там свою систему сообщения. Лифтом служит унитаз, а ключом - туалетная бумага, причем, где бы ты ни сел в лифт-унитаз, тебя всегда выбросит на месте старта - в подвале. Всего имеется два унитаза и три рулона бумаги.

В некоторых комнатах нет освещения. В некоторых из пола торчат острые колья. Практически везде ползают пауки. В общем, на каждом шагу меня поджидает смерть. И я начал было уже сетовать на малое количество дополнительных жизней, когда оказался во дворце.

Первым делом я огляделся и заметил небольшое послание на стене. Так как я видел его и раньше, то проигнорировал его и пошел из подвала направо. Тут я наткнулся на ядовитых пауков, прыгавших туда-сюда. Неприятности добавляли падающие с потолка плиты. Стремительно пробежав эту зловещую комнату, я попал в другую, не менее ужасную - здесь тоже были пауки, а из пола торчали острые колья. Зато здесь была и нужная мне туалетная бумага. Я взял ее, поднялся по плитам наверх и стал обладателем магического напитка. Какой-то бедняга, бродивший здесь до меня, оставил его и, по-видимому, уже никогда за ним не вернется.

Вернувшись на место старта, что стоило мне немного энергии, я поднялся выше, потом еще выше, вошел в комнату, где ползала улитка и взял там мешок с крупой. Так как больше трех предметов одновременно я поднять не могу, то мне стало тяжеловато. Спустившись вниз, я прошел налево через картинную галерею с одной-единственной картиной и добрался до входа на второй этаж. Но лестницы не оказалось - сломалась, наверное. Зато там был котел, в котором варилось какое-то варево.

Осторожно миновав улиток и ядовитых ос, я запрыгнул на котел и положил в него крупу и магический напиток. Не знаю, чего я от этого ждал, наверное ничего, просто устал их носить на себе. Так оно и вышло - ничего и не произошло. Разочарованно вздохнув, я отправился дальше.

Вернувшись в ту комнату, где я брал туалетную бумагу, я пошел направо. Перепрыгнул колья, обошел пауков и очутился в новом месте. Здесь была банка с клубничным джемом. Когда-то я очень любил этот джем, но сегодня брать его не стал. Решил, что прихвачу на обратном пути и правильно сделал. Начались такие неприятности, что мне стало не до джема.

Сначала я упал в шахту и при этом чуть не наступил на ядовитых крыс. Со страха сунулся было налево, но там была такая темнота, что идти дальше было бы чистым безумием. Пришлось собраться с духом и осторожно обходить крыс. Так я оказался в комнате с рыцарскими доспехами. Здесь мне наконец повезло - я нашел фонарик. Поспешно схватив его, я бросился в темную комнату и осветил мрачную обстановку заброшенного подвала.

Луч фонарика выхватил из темноты старую засушенную рыбу. Прихватив ее на всякий случай, я осторожно пошел дальше. В соседней комнате меня ожидали бутылка с соком и целый ряд острых кольев. Так как сок я все равно взять не мог, то со спокойной душой проследовал дальше, в комнате со статуей были плиты, а чуть ниже - пауки. Осторожно обойдя их, я очутился в зловещей комнате с двумя проломами в полу. Между проломами прыгал паук и лежала кассета. Логически поразмыслив, я подумал, что кассета мне вроде бы ни к чему. Оставалось падать в один из проломов. Я выбрал дальний и, как оказалось, - не напрасно.

Недолгое падение в шахту, и я очутился в самом низком месте замка. Здесь лежала туалетная бумага. Она у меня уже есть и новую я брать не стал, а решил запомнить этот путь, чтобы вернуться за ней в случае необходимости. Поскольку больше идти было некуда, я пошел налево и увидел... унитаз. Я уже упоминал о двойном назначении этого замечательного устройства. Подойдя к нему, я запрыгнул на сиденье, дернул за ручку и меня доставили на место старта. Бумага при этом у меня исчезла, а рыба и фонарик - остались.

Отнеся рыбу к котлу, я вернулся известным путем за бутылкой сока. Вооружился туалетной бумагой, воспользовался унитазом, оказался на месте старта и положил сок в котел. На этот раз мне повезло, в котле все забулькало, забурлило и я обрел возможность ЛЕТАТЬ!!! Взлетев, я оказался на втором этаже.

* 2 этаж *

Я сразу пошел налево. Осторожно обойдя в следующей комнате пауков и увернувшись от падающих плит, я нашел непонятный глаз, оставшийся здесь от какого-то доисторического чудовища. Поднявшись по плитам наверх, я увидел карликовую собачку, но брать ее не стал. Так я пробирался все выше и выше, стараясь при этом держаться правее. Дорога привела меня на крепостную стену. Прихватив невесть откуда взявшийся там гамбургер и увернувшись от летучих мышей, я снова спустился в замок.

В комнате, левее моей, я увидел цель своего путешествия - спящего короля. И здесь меня осенила идея. Моя месть будет еще более страшной, если я разбужу его с помощью будильника. Насколько я помнил, в мире живых это устройство считалось самым кошмарным после бормашины. Где взять бормашину я не знал, а вот будильник в замке когда-то был. Я сам устраивал его веселые похороны на старом кладбище, развлекая короля и гостей.

Пойдя направо, я перепрыгнул улитку и оказался в комнате с рыцарем. Там тоже было какое-то послание, но читать его мне было некогда. Я поспешно спустился вниз к котлу, положил в него глаз и гамбургер, вернулся на дворцовую стену и взял там баллон с кислородом.

Возвратившись обратно, я не стал пробираться к котлу, а спустился в шахту на одну

комнату вниз. Здесь я стал счастливым обладателем вкусной конфеты. Только теперь я пошел к котлу. Выложил в него и баллон и конфету, но ничего не произошло. Продолжив поиски, я нашел этажом выше воздушный шарик. С ним я направился к выходу на первый этаж - в ту комнату, где стоит первый котел. Но как я ни запрыгивал на него в надежде пробраться дальше в глубины замка, меня всегда относило наверх. Тут мне пришлось расстаться с одной моей дополнительной жизнью. До сих пор думаю, что что-то я сделал не так. Хоть все и кончилось хорошо, но гложет червячок сомнения мою неуспокоенную душу.

Очнулся я в картинной галерее. Обрадовавшись, побежал к комнате с доспехами, где в прошлый раз брал фонарик. Упав вниз, я взял третий рулон туалетной бумаги, перепрыгнул паука и проследовал дальше.

В комнате, куда я упал, было два хода - направо и налево. Пройдя налево, я увидел второй "лифт", но уезжать пока не стал, т.к. помнил, что меня выбросит на место старта. Поэтому пошел направо. Там и обнаружился подводный ход, ведущий наружу, о котором говорилось в древних легендах. Я смело прыгнул в холодную воду и не утонул, благо у меня был шарик. Немного поплавав, я выбрался из воды на старом кладбище вне пределов замка. Побродив по нему, я наконец обнаружил за мраморными статуями долгожданный будильник, заведенный на три часа утра…

Остальное было делом техники. Я без осложнений пробрался сквозь стаи голодных пираний, использовал унитаз, очутился на месте старта, поднялся на второй этаж, добрался до спальни короля и взобрался на кровать. Потом я зловеще расхохотался и нажал на кнопку будильника. Вы бы видели, какие в тот момент у короля были глаза!!!

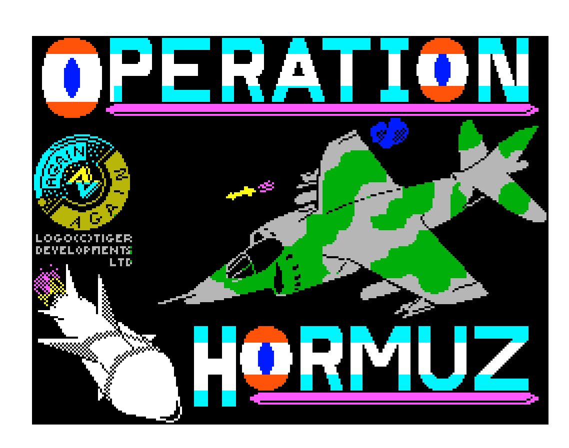
Так закончилась эта смешная и грустная история, приключившаяся со мной совсем недавно. До новой встречи друзья!

СОВЕТЫ ЭКСПЕРТОВ

Сегодня этот раздел игровых программ полностью посвящен авиаимитаторам. Все описания подготовлены одним автором - это наш постоянный корреспондент из города С.-Петербург - Фокин С.А.

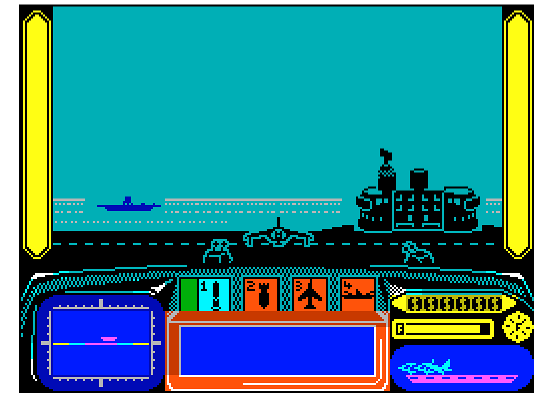
OPERATION HORMUZ

"Durell" 1988г.
Эксперт Фокин С.А.
г. С.-Петербург



В этой игре Вы будете управлять легким штурмовиком с вертикальным или укороченным взлетом и посадкой - AV-8A "Харриер", который используется в военно-морских силах Великобритании. Вам предстоит очень много стрелять, бомбить, пускать ракеты, и все это для успешного выполнения операции "Хормуз", цель которой - уничтожение ракетных баз противника, а точнее - его шахт, в которых базируется стратегические ракеты.

Имитация полета выполнена довольно условно: Ваш самолет летает как бы на плоскости, а Вы управляете им, глядя со стороны.



Запускается игра нажатием на "1", а управление осуществляется стандартными клавишами, хотя у Вас будет возможность их переназначить. После старта на экране Вы увидите свой авианосец с самолетом на борту. Слева и справа от основного экрана расположены индикаторы повреждений самолета. Внизу расположена приборная панель, у которой слева находится радар, посередине - экран предупреждений и сообщений, над которым находятся указатели различных типов оружия. Указатель выбранного оружия включен всегда. В правой части приборной панели находятся: счетчик очков, индикатор горючего и количество оставшихся самолетов.

Клавиши "1","2","3","4" - выбор оружия.

Самолет управляется клавишами:

"A", "Z" - вверх, вниз (управление по тангажу);

"N","M" - наклон по и против часовой стрелки (управление углом крена);

"SPACE" - стрельба выбранным оружием;

"F" - запуск тепловой ракеты /FLARE/;

"Q" - окончание игры.

Нажав "S", Вы можете наблюдать на карте вражеские базы, расположение вашего авианосца и количество оставшихся самолетов.

Итак, вы произвели взлет, оторвались от палубы и полетели направо к 1-ой вражеской

базе. Время от времени Вам будет выдаваться сообщение, что впереди находятся вражеские корабли. Они оснащены противокорабельным ракетным комплексом "Экзосет" и если Вы не хотите по возвращении обнаружить свой авианосец затонувшим (CARRIER SUNK), то уничтожайте их немедленно, иначе будьте уверены - начнется обстрел авианосца и будут попадания (CARRIER HIT).

Другая постоянная угроза - истребители МИГ-21 советского производства. Они атакуют ракетами "воздух-воздух" с инфракрасным наведением. Чтобы избежать попадания, необходимо произвести пуск тепловых ракет, которые уведут самонаводящиеся ракеты от Вашего самолета. Компьютер предупреждает об этом сообщением "LAUNCH FLARE".

О приближении к ракетной базе предупреждает сообщение - "BASE1". Тут Вам придется показать все свое искусство в заходе на цель, бомбометании и маневрировании, т.к. ракетные шахты (они все имеют квадратную форму) можно поразить только бомбами. Когда Вы разрушите все шахты, компьютер выдаст сообщение "DESTROYED". По мере того, как Вы будете удачно продвигаться по территории врага, Ваш авианосец будет двигаться вслед за Вами. Так что для дозаправки и пополнения боеприпасов не нужно будет далеко лететь, на карте боевых действий свое местоположение Вы можете определить по мигающему номеру базы: если мигают 2 номера, значит Вы находитесь между базами.

Операция будет успешно завершена, если Вы очистите всю территорию от ракетных баз противника.

ACE

"Cascade", 1986 г.
Эксперт Фокин С. А.
г. С.–Петербург



"ACE" - это типичный имитатор воздушного боя с довольно упрощенной техникой пилотирования, но с разнообразными типами вооружения. Отличительная черта боя - высокая агрессивность.

Агрессор предпринял попытку вторжения на Вашу территорию как сухопутными, так и военно-морскими силами. Все его боевые операции сопровождаются мощной поддержкой со стороны авиации. Вам предстоит нелегкая задача отразить нападение врага и защитить суверенитет своего государства.

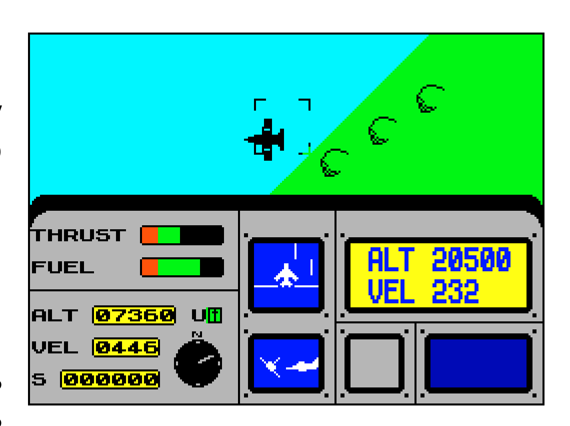
После загрузки нажмите "SPACE" и на экране появится шифр. Его можно корректировать клавишами "5" и "6". Когда будете готовы, нажмите "SPACE" и компьютер запросит коды. До Вас здесь уже поработали хаккеры, так что подходит довольно много вариантов из 2-х букв, цифр или символов. Наберите, например, "12" или "88", а можете подыскать какие-то свои варианты, после этого на экране появится основная заставка, на которой Вам покажут различные вражеские объекты, встречающиеся в ходе сражения. При дальнейшем нажатии любой клавиши Вы переходите в основное меню:

1 - старт игры
2 - уровень сложности
3 - выбор пилота-одиночки или пилота со стрелком
4 - выбор летных условий (лето, зима, ночь)

5 - загрузка таблицы лучших результатов
6 - показать таблицу лучших результатов
7 - включение/выключение кемпстон-джойстика

После старта появится небольшое меню, в котором Вы можете сценарий и полетное задание:
1 - многоцелевой вариант;
2 - цель - воздушное превосходство;
3 - поддержка сухопутных операций;
4 - поддержка военно-морских операций.

Затем вы оказываетесь на взлетной полосе одной из ваших баз. Управление самолетом очень простое:
"CAPS SHIFT", "Z" - обороты двигателя;
"E","R" - влево, вправо;
"W", "S" - вверх, вниз;
"X" - "огонь";
"ENTER" - выбор оружия;
"U" - шасси;
"M" - карта;
"Q" - окончание игры.



Приборная панель имеет в центре радар и указатель угла крена самолета. Слева вверху находятся индикаторы оборотов двигателя и горючего. Слева внизу - указатели высоты и скорости, а также индикатор положения шасси и указатель курса. Здесь же находится счетчик очков.

В правой части панели расположены: вверху - табло сообщений, внизу - камера заднего вида и указатель выбранного оружия.

Теперь Вы можете непосредственно начать сражение. Чтобы взлететь, необходимо набрать скорость не менее 150 миль в час, но не злоупотребляйте нагрузкой на шасси и не забывайте, что после взлета его следует убрать.

Определите по карте ближайшее сосредоточение вражеских сил, и немедленно направляйтесь в зону боевых действий. В зависимости от того, как быстро Вы будете обнаруживать и уничтожать цели, во многом и зависит исход сражения.

Наземные цели можно уничтожать как из пулемета, так и ракетами "воздух - земля". Большое неудобство доставляют вертолеты противника. У них есть средства защиты от Ваших ракет, и лучше их расстреливать из пулемета. Но главная помеха - это вражеские самолеты. Истребители летают звеньями (2 самолета); количество нападающих звеньев определяется уровнем сложности игры.

Истребители непрерывно атакуют ракетами, о чем выдается предупреждение - "MISSILE WARNING". Чтобы избежать попадания, надо вовремя выпустить уводящую помеху - "DECOY FLARE" и когда ракета будет уведена от Вашего самолета, вы получите сообщение - "MISSILE AVOIDED", в противном случае, после какого-то количества попаданий (DAMAGED SUSTAINED), у Вас могут быть повреждения.

Истребители противника уничтожить очень непросто, но это надо сделать, чтобы до подхода очередной волны авиации, успеть провести подавление сухопутных сил на каком-либо фронте.

Время от времени на высоте 20500 футов будет появляться дозаправщик (REFUELER). Чтобы пополнить горючее в воздухе, необходимо набрать его рабочую высоту, зайти ему в хвост, подлететь как можно ближе и совместить конец всасывающего устройства с заправочной форсункой. Не забывайте, что скорости при этом должны быть одинаковыми.

У Вас в резерве есть один головокружительный прием: нажав клавишу "J" в любой

момент сражения, Вы окажетесь на взлетной полосе одной из Ваших баз. Но, к сожалению, за всю игру Вы не можете нажать ее более 3-х раз.

Противник будет стремиться уничтожить ваши взлетные полосы, и, если Вы плохо ведете бой, то можете получать сообщения: "AIR FIELD DESTROYED". Когда все Ваши взлетные полосы будут разрушены, то считайте, что вы проиграли, т. к. Вам просто негде будет пополнить горючее и боеприпасы. Итак, желаем удачи в боевой пилотировании.

ACE - 2

"Cascade", 1987 г.
Эксперт Фокин С. А.
г. С.-Петербург



Игра "ACE-2" является имитатором воздушного боя, в котором основную роль играет правильное оснащение Вашего самолета.

Сюжет "ACE-2", в отличие от игры "ACE", довольно прост: есть наземная цель противника, есть прикрытие со стороны авиации. Надо уничтожить и то и другое. Время от времени предстоят возвращения на базу для смены систем вооружения, пополнения боекомплекта и для дозаправки горючим.

После загрузки программы нажмите клавишу "SPACE", и на экране появится обширное меню. В него входят следующие пункты:

- начать воздушный бой:
- выбор противника (партнер или компьютер);
- сценарии сражения:

1 - ближний бой - оба самолета уже находятся в воздухе и вооружены только ракетам малого радиуса действия;

2 - полная имитация боевых действий по схемам "воздух-воздух" и "воздух-земля";

- количество самолетов для каждой стороны (от 1 до 20); высвечивание места аварии;
- количество попаданий ракет для уничтожения самолета;
- выбор джойстика.



Курсор перемещается клавишами "Q" и "A", изменение пункта производится клавишей "F". Если в меню выбран синклер-джойстик, то им можно пользоваться одновременно с клавиатурой.

Выбрав параметры боя и переместив указатель на первый пункт, Вы начинаете игру. Перед тем, как непосредственно начать бой, Вам будет предоставлена возможность выбора вооружения.

Если Вы играете против компьютера, то Ваше меню находится в верхней части экрана. Манипулируя курсором также, как и в основной меню, Вы можете выбрать необходимые Вам типы ракет и после этого идти на взлет.

В нижней части экрана расположена приборная панель, в центральной части которой

находятся радар, авиагоризонт, а между ними - указатель угла тангажа.

В левой части панели находится экран, на котором внизу указан выбранный тип оружия и его комплектность, а вверху - высвечиваются сообщения и предупреждения.

В правой части панели находятся индикаторы уровня топлива и мощности двигателей, а под ними - указатели скорости и высоты самолета. В самом правом конце расположен указатель курса.

Ваша задача - уничтожение вражеских самолетов, а также ликвидация наземной цели противника. В то же время, надо оберегать свой корабль в случае попытки потопить его. Расположение всех объектов в любой момент можно увидеть, включив карту (клавиша "C").

Органы управления программой:

"Q", "A" - угол тангажа;
"Z", "X" - угол курса;
"F" - "огонь";
"S", "D" - обороты двигателей;
"W" - выбор оружия.

Надо отметить, что после пуска ракеты по цели на экране появляется сообщение "TARGET LOCKED" - "цель захвачена" и, пока ракета не достигнет цели, вторую выпустить невозможно, на каждый вылет самолет может взять ракеты только 2-х типов, поэтому периодически придется возвращаться на базу.

Чтобы вернуться на базу, необходимо подлететь к левому краю карты и иметь при этом высоту менее 1000 футов.

Самолет может быть вооружен ракетами трех типов:

"HEAT AIR-AIR" - ракета класса "воздух-воздух" с тепловой головкой самонаведения.

"RADAR AIR-AIR" - ракета "воздух-воздух", с наведением по отраженному лучу подсвечивающего радара.

"AIR GROUND" - ракета класса "воздух-земля".

Пуск ракет можно осуществить только тогда, когда система наведения "захватит" цель. Об этом свидетельствует появление придела на лобовом стекле. Ракеты 2-го и 3-го типа требуют "подсветки" цели. Поэтому надо сопровождать цель и после пуска ракеты, т.е. держать ее в области прицела. Ракета с тепловой головкой самонаведения дойдет до цели сама, хотя ее недостатком является малый радиус действия - до 10 миль.

Необходимо отметить, что наземная цель противника надежно защищена системой ПВО, и Вас собьют раньше, чем Ваша ракета достигнет цели. Чтобы этого не произошло, Вам необходимо ставить радиопомехи. Они собьют с курса ракеты ПВО и не позволят уничтожить Вас и Ваши ракеты.



Чтобы уничтожить наземную цель, необходимо попадание 2-х ракет. Радиопомеха ставится клавишей "R", о чем свидетельствует сообщение - "CHAFF LAUNCH". Для этой же цели служат тепловые активные фальшцели (FLARE), но они отводят только ракеты с тепловой головкой самонаведения. Они запускаются тоже клавишей "R".

В заключение дадим ряд полезных советов. У Вас есть сильный козырь - это вид из кабины самолета противника, и Вы всегда сможете увидеть выпущенную в Вас ракету. От ракеты, ведомой по отраженному лучу, можно уйти, совершив маневр и выйдя из луча подсвечивающего радара. Если вы не уверены, чья ракета выпущена раньше Ваша или противника, то советуем Вам после пуска снизить скорость.

При игре вдвоем конфигурация приборной панели 2-го самолета отличается от первой, но ее нетрудно освоить, т.к. она содержит все те же элементы.

Клавиши управления второго самолета следующие:

"P","L" - угол тангажа;

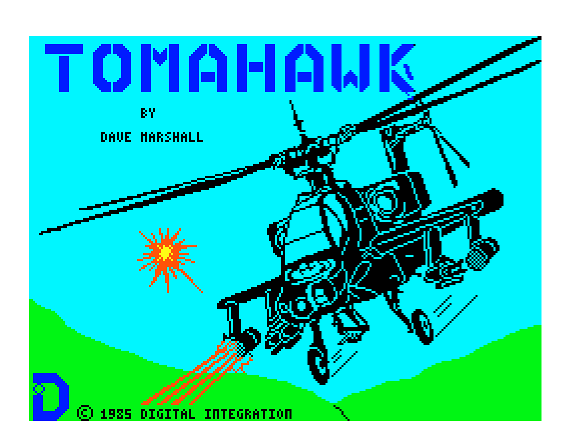
"N","M" - угол курса;
"H" - "огонь";
"O" - выбор оружия;
"J","K" - мощность двигателей;
"U" - постановка помех.

Перевод важнейших сообщений:

"ENEMY PLANE" - вражеский самолет
"RANGE" дистанция
"ALT" - высота
"ENEMY MISSILE" - вражеская ракета
"OUT OF RANGE" - вне зоны досягаемости
"CRASH DANGER" - угроза аварии
"STALL DANGER" - угроза вхождения в "штопор".
"MISSILE LAUNCH" - пуск ракеты

TOMAHAWK

"Digital Integration" 1985 г.
Эксперт Фокин С.А.
г. С.-Петербург



Игра "Томагавк" - это один из самых качественных имитаторов боевого вертолета. Техника управления и полета выполнена на очень высоком уровне и максимально приближена к реальности.

Идет война, силы противника перевешивают. Командование объединенных сил решило применить последний шанс - операцию под кодовым названием "Томагавк", в основу которой положена поддержка наземных операций с воздуха с помощью вертолетов огневой поддержки типа "Апач". Вам и предстоит управлять одним из таких вертолетов.

После загрузки нажмите 2 раза "ENTER", и на экране появится меню настройки:

1 - выбор вариантов сражения (см. ниже)
2 - день или ночь
3 - чистое небо или облачно
4 - высота облачности
5 - ветренность и турбулентность
6 - включение/выключение звука
7 - рейтинг пилота (курсант, новичок, инструктор, ас)
8 - выбор управления.

После настройки нажмите "ENTER", игра начинается с того, что Вы находитесь на посадочной площадке. Для начала необходимо освоить управление:

Клавиши "W", "S" увеличивают и уменьшают обороты двигателя;
"Q", "A" - изменяют крутящий момент на винте.

Если вертолет висит на месте или имеет малую скорость, то "CAPS SHIFT" и "Z" разворачивают его.

Чтобы вертолет набрал скорость, надо наклонить его вперед (клавиша 7), но не забудьте набрать достаточную высоту. Аналогично можно погасить скорость, наклонив его назад (клавиша 6).

Клавиши 5 и 8 управляют креном вправо и влево.

Зону боевых действий в любой момент можно увидеть, нажав клавишу "M".

Клавишей "C" осуществляется наведение на различные типы объектов:

"H" - посадочная площадка:

"B" - база;

"T" - танк или орудие:

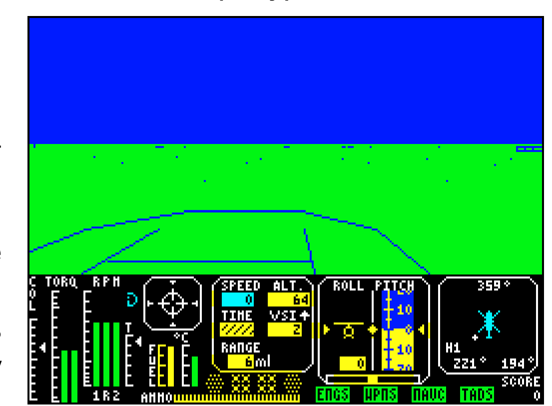
Две стилизованные буквы "S" -это вертолет.

Если целей выбранного типа несколько, то все их можно просмотреть клавишей "N". При выборе в качестве цели вертолета, танка или орудия, на лобовом стекле появляется прицел. Каждому виду оружия соответствует прицел определенной конфигурации:

"X" - пулемета;

"Крест" - для неуправляемых ракет;

"Квадрат" - для управляемых ракет.



Выбор оружия осуществляется клавшей "P", а стрельба из него клавишей "O".

Игра может быть остановлена клавиши "H" и продолжена клавишей "J". Одновременное нажатие "CAPS SHIFT" И "SPACE" - окончание игры.

Приборная панель вертолета расположена в нижней части экрана и достаточно сложна, поэтому требует отдельного описания.

В левой части экрана панели находятся вертикальные индикаторы крутящего момента винта (TORQ), оборотов двигателей (RPM), горючего (FUEL) и температуры системы охлаждения (C). Над индикаторами горючего и температуры расположен экран захвата цели, по которой можно произвести выстрел.

В центре панели находятся указатели скорости, высоты, скорости изменения высоты, времени полета до выбранной цели при установленной скорости, а также указатель расстояния до выбранного объекта.

В правой части панели расположены два экрана: левый показывает тангаж и крен, а правый - курс полета вертолета, азимут цели, тип и номер выбранной цели. В самом низу панели наводятся данные о боеприпасах и несколько табло повреждений вертолета (двигатели, вооружение, навигационные приборы и хвостовое оперение).

Как уже было упомянуто, цель игры - очищение каждого квадрата зоны боевых действий (на карте они мигают) от танков и орудий врага.

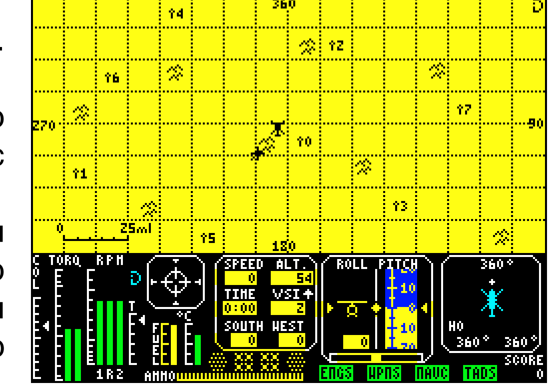
В зависимости от выбранного в основном меню варианта сражения, у Вас будут немного отличающиеся задачи.

1. Тренировочный полет: вам предоставляется возможность летать, стрелять, но боевые действия между войсками не ведутся

2. Все пространство поделено на две части. Сражение идет в средней полосе линии фронта.

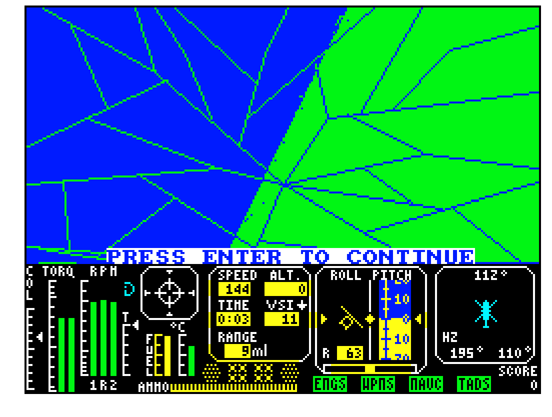
3. Все оперативное пространство занято противником, и Вам нужно очищать область, начиная с центрального квадрата.

4. Этот вариант подобен варианту 2, но бои ведутся по всей линии фронта. Все пространство условно разбито на 128 квадратов, так что Вам потребуется огромное терпение, чтобы довести начатое дело до конца.



Цифрами на карте обозначены базы. На них Вы можете произвести ремонт в случае

повреждения оборудования. Пополнение боеприпасов происходит на посадочных площадках. Следует знать, что при возникновении аварийных условий, вам выдается звуковое предупреждение, и, чтобы избежать катастрофы, необходимо как можно быстрее вывести вертолет на нормальный режим полета. Например, при пикировании на цель, у Вас может развиться скорость более 200 миль/час. Если вы не погасите скорость, то у вас попросту оторвутся лопасти.



В заключение следует напомнить, что Вы можете навестись на цели только в пределах квадрата, в котором находитесь, и не пытайтесь сесть на базу, если она находится в квадрате, захваченном врагом. Если противник прорвал фронт и завоевал все квадраты от левого края до правого, то Вы уже не сможете отвоевать эту линию обратно.

Танки можно подбить только ракетами.

Наиболее удобно сбрасывать скорость быстрыми поворотами вправо-влево.

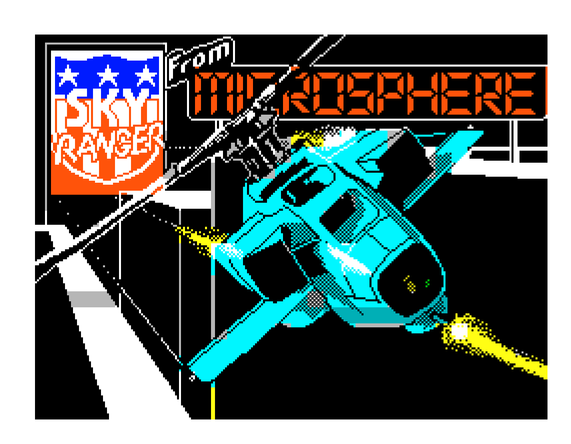
Вертолеты противника появляются по-одному в том квадрате, где Вы находитесь.

Ваши ракеты имеют дальность действия 3 мили.

На этом можно поставить точку, а все остальное вы обнаружите и прочувствуете сами. Успеха вам!

SKY RANGER

"Microsphere", 1985г.
Эксперт Фокин С. А.
г. С.-Петербург



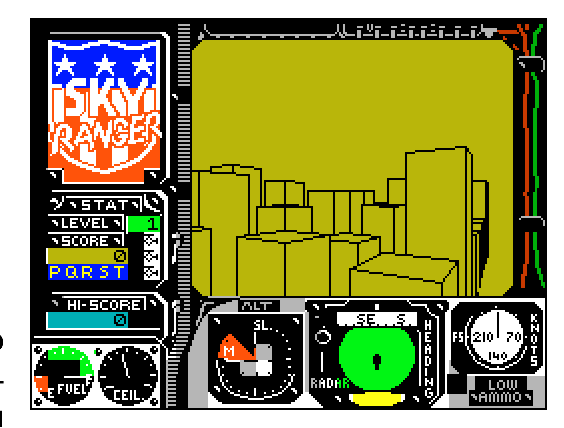
Игра "SKY RANGER" - это упрощенный имитатор полета на вертолете с элементами погони и стрельбы. Полет проходит в очень динамичной форме (особенно на высоких уровнях), поэтому игра захватывает на длительное время.

В современном городе на севере Америки появились загадочные объекты в виде черных шаров до 1 метра в диаметре. Они с поражающей быстротой пожирают атмосферный кислород, но не брезгуют также и людьми. Правительство поручило очистить город от этой заразы Вам - военнослужащему воздушного диверсионно-разведывательного подразделения.

После загрузки, если не нажимать на клавиши, начнется демонстрация полета. При нажатии любой клавиши компьютер запрашивает: будете переназначать клавиши? (да/нет). После этого необходимо ввести пароль уровня сложности.

При нажатии на "ENTER", запускается 1-й уровень. Перед Вами вид из кабины вертолета. Приборная панель проста: справа - указатель скорости, под которым находится табло наличия боеприпасов, в центре панели - радар с компасом, под радаром расположен индикатор захвата цели. Слева от радара находится высотомер. В левой части панели

расположены указатели горючего и высоты облачности.



Управление от стандартных клавиш следующее:
"CAPS SHIFT" ... "V" - поворот влево;
"B" ... "SPACE" - вправо;
"A" ... "G" - уменьшение скорости;
"Q" . . . "T" - увеличение скорости;
"H" ... "ENTER" и "Y"... "P" - изменение высоты;
Весь верхний ряд - "огонь".

Чтобы перейти на следующий уровень, Вам надо сбить 16 шаров. На каждый уровень Вам дано 4 вертолета. Чем выше уровень, тем сложнее догонять и сбивать шары, т.к. их скорость возрастает. По шару можно произвести выстрел, когда мигает индикатор под радаром. Для этого шар должен быть по высоте немного ниже средней линии лобового стекла.

Если вы внезапно попали в облачность, снизьте скорость и опуститесь ниже.

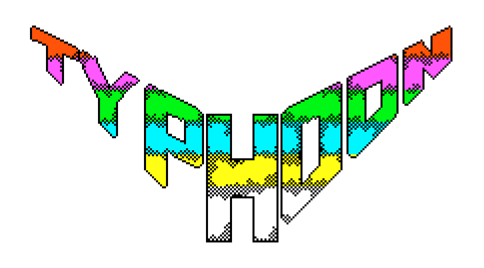
Сажайте вертолет плавно, т.к. иначе на стекле будут трещины, которые впоследствии будут Вам мешать.

Если горючее кончается, то лучше всего сесть на землю и подождать, когда оно кончится окончательно. При этом вы лишаетесь вертолета, но зато у Вас не будет трещин.

При переходе на следующий уровень компьютер выдаст Вам его код. Не забудьте записать его или запомнить, чтобы в следующий раз не начинать сначала. Хочется надеяться, что Вы оправдаете надежды жителей города.

TYPHOON

"Ocean", 1988г.
Эксперт Фокин С. А.
г. С.-Петербург



"TYPHOON" - это аркадная многоуровневая игра, которую можно отнести к разряду "боевиков" ("Action"). На всех отдельно загружаемых уровнях Вам придется вести активные боевые действия с воздуха, в самой разнообразной боевой обстановке, встречаясь с невиданным огнем противника, используя различные виды оружия. Вы должны добиться победы. В противном случае - смерть!

Отличная графика, единая связь всех уровней в общем контексте игры, и в то же время их разнообразие - вот, что Вас ожидает.



Итак, после загрузки основного блока надпись вверху начнет мерцать. Выберите нажатием соответствующей клавиши Ваш вариант управления (никакой реакции на экране не будет), а затем нажмите клавишу "N". После этого Вы можете загрузить 1-й уровень, после загрузки появится предупреждающая надпись, после чего вы оказываетесь непосредственно в игре.

Задача 1-го уровня - уничтожение авианосца противника. Ваш самолет будет снижаться сквозь облака, встречая яростный огонь врага. А после снижения окажется непосредственно над авианосцем. Произведя 3 метких выстрела в машинное отделение

(это около трубы), Вы сможете насладиться видом разрушенного корабля. На этом уровне Вам понадобятся клавиши:

"Q","A" - вперед, назад;
"O","P" - влево, вправо;
"N" - "огонь".

Надо отметить, что конфигурация игры не очень удобная: если Вы не прошли какой-либо уровень (даже 1-й), Вам придется опять произвести загрузку с 1-го уровня, впрочем это недостаток многих программ, имеющих подзагружаемые уровни сложности.

2-й уровень уже сложнее 1-го. Ваша задача пролететь над всеми оборонительными укреплениями, уничтожая как можно больше на своем пути. Когда укрепления закончатся, картинка на экране остановится, и Вам нужно будет уничтожить главную цель в верхней части экрана. На этом уровне выбор оружия более разнообразен: наземные цели можно уничтожать бомбами (клавиша "B"). Иногда Вам могут попадаться контейнеры с буквой "E". Это батареи для зарядки лазера. Если их перехватить, то вместо скорострельной пушки "VULCAN", будет производиться стрельба лазером.



В списке вооружений, который находится на экране слева, всегда указывается задействованный тип оружия. С потерей "жизни" теряется то, что Вы приобрели. Теперь о самой интересном: вы, конечно, обратили внимание на изображение в нижней правой части экрана. Это сверхмощная ракета-аннигилятор, которая уничтожает все в пределах видимости. Она дается только одна (на каждую жизнь) и запускается клавишей "M".

3-й уровень отличается от предыдущего только ландшафтом и конечной стратегической целью. Здесь Вы также сможете использовать более широкий выбор оружия.

По-видимому, нет смысла подробно описывать оставшиеся уровни. В конце каждого из них вас ждет загадочный объект врага, который Вам предстоит уничтожить. После уничтожения всех объектов, на экране появится пилот, который сообщит об удачном выполнении миссии.

Желаем удачи!

TOP GUN

"Ocean", 1986г.
Эксперт Фокин С.А.
г. С.-Петербург



Игра "TOP GUN" представляет собой нечто среднее между имитатором полета истребителя и боевиком жанра "ACTION".

Для того, чтобы быть чистым имитатором, она слишком проста в управлении, не думаю, чтобы кто-то именно так представлял себе управление современным истребителем. Для обычного "ACTION" она все-таки сложновата: требует каких-то представлений о полете и маневрировании в воздухе. Но, как бы то ни было, именно такое сочетание и оказалось весьма удачным с точки зрения игрового впечатления.

Термин "TOP GUN" обозначает школу асов-истребителей военно-морских сил США. Туда принимают самых лучших пилотов ВМС и делают из них истинных мастеров воздушного боя. Если будет возможность посмотреть одноименный фильм, обязательно посмотрите - не пожалеете (у нас фильм известен под названием "Воздушная гвардия" - Прим. "Инфоркома"). Увидев заставку этой игры, вы сразу же узнаете (или не узнаете) изображенный самолет - это 2-х местный тяжелый истребитель морского базирования F-14 "Томкэт". Он приспособлен для взлета и посадки на авианосцы и действует в основном именно с таких кораблей.

Сражаться придется, как ни странно, против однотипного самолета. Это не то маневры (странно, когда пораженный самолет взрывается), не то враг обладает такими же машинами. Впрочем, если кому-то захочется, то он может вообразить себе, что сражается на СУ-27 морского базирования, которые взлетают с авианосца типа "Кремль", против вражеских F-14.

Во время воздушного боя в вашем распоряжении - пушка, ракеты, радиолокационные ловушки и, конечно, маневр. Для выполнения 1-й миссии достаточно сбить 3 самолета противника. Вражеские летчики в 1-й миссии маневрируют весьма слабо и плохо умеют использовать оружие. Они всегда стреляют из пушки и не применяют ни ракет, ни радиолокационных ловушек против Ваших ракет. Учитывая, что с помощью пушки самолет сбить достаточно сложно (необходимо много попаданий), то лучше использовать ракеты. Для этого необходимо, чтобы самолет противника несколько секунд удерживался в центре прицела.

На ракеты Вы должны переключаться заранее с помощью клавиши переключения оружия. При этом прицел имеет форму квадрата. Головка самонаведения должна захватить цель и, если теперь нажать клавишу "огонь", то вы увидите удаляющуюся ракету, которая сама наводится на самолет противника. Если самолет противника находится слишком близко от Вас, то сколько бы мы ни держали его в центре прицела, захвата цели не происходит. Так что оружие это очень эффективное и капризное одновременно. Поэтому во время выполнения миссии 2 (MISSION 2) советуем Вам пользоваться преимущественно пушкой, т.к. стоит только пустить ракету, как опытный враг тут же пускает ловушку и ракета сбивается с правильного курса. При этом вражеский ас, пока Вы возитесь с ракетным прицелом, успевает всадить в вас две-три очереди из своей пушки.

Как ни странно, противник почти не пользуется ракетами. Во 2-й миссии асы противника воюют на порядок лучше, чем в 1-й. Сбить их очень трудно, они выписывают немыслимые пируэты, когда сближаются с Вами и все время стремятся сесть Вам на хвост. Стряхнуть их очень трудно, здесь может выручить только мастерское маневрирование.

Несколько полезных советов по маневрированию во время боя:

Чтобы развернуть самолет, надо выполнить поворот самолета вправо, либо влево и при этом изменять высоту полета. При этом, чтобы ориентироваться и контролировать выполнение разворота, полезно наблюдать за самолетами противника на радаре: Вы должны стремиться, чтобы из нижней части круга они попали в верхнюю.

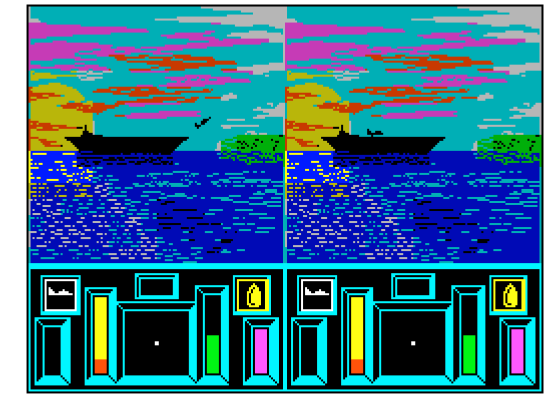
Во время боя полезен также маневр по высоте. Однако, следует иметь в виду, что при крутом пикировании высота падает очень быстро, а при достижении значения 000 Вы сами знаете, что будет с вашим самолетом. При крутом наборе высоты следует следить за тягой двигателя: если она недостаточна, самолет начинает терять скорость и может сорваться в штопор.

Вообще-то вы уже, видимо, заметили необычную особенность этой игры: на экране вид из двух кабин самолетов: вашего и вражеского. Такое встречается довольно редко и здорово облегчает жизнь, можно не только видеть вражеский самолет, но и наблюдать, как враг видит Вас. Это помогает ускользать от вражеского прицела. Полезно также во время боя смотреть на приборы противника и поучиться у него некоторым приемам. Особенно обратите внимание на то, как часто он меняет тягу двигателя – это позволяет ему зайти Вам в хвост.

Приборы помогут Вам контролировать обстановку. Вверху от основного экрана находятся высотомер (в футах) и индикатор мощности двигателя. Под основным экраном расположен радар. В его центре жирной точкой обозначен Ваш самолет, а маленькой - самолет противника. Интересно, что стоит только сбить вражеский самолет, как тут же появляется другой. Они всегда появляется сзади или сбоку. Так что до того, как они сблизятся с Вами, не плохо было бы развернуться.

Разберемся дальше с изображением в нижней части экрана. Верхний квадрат слева с силуэтом самолета показывает положение относительно горизонта. Квадратик справа содержит стилизованное изображение включенного оружия. При переключении оружия меняется вид прицела на основном экране. Радиолокационные ловушки прицела не имеют, средняя полоска внизу показывает количество повреждений самолета, а полоска справа - тяга двигателя.

После загрузки программы появится меню, в котором можно выбрать количество игроков (необходимо нажать 1 или 2). В общем-то, на начальной стадии игры полезно выбрать режим 2-х игроков и играть одному. При этом вражеский самолет - неуправляем, он не маневрирует и не стреляет. Идеальная мишень для начинающего пилота!



Чтобы войти сразу в игру (если управление устраивает), то надо нажать клавишу "SPACE", а если нет, то "ENTER". При этом появится новое меню для 1-го игрока. Курсорными клавишами можно выбрать нужную опцию и, если выбрать больше нечего, нажать "SPACE" или "ENTER", если надо выбрать управление для второго игрока.

При назначении клавиатуры появляется соблазнительная надпись: "PRESS D TO DEFINE KEYS". При этом идти на поводу у компьютера не следует: если переназначить клавиши, то игра зависнет на картинке с авианосцами, ничего не поделаешь, так уж с ней поработали компьютерные хаккеры. Дело в том, что эта игра в оригинальной версии работала только на фирменных "ZX SPECTRUM", а на наших моделях она зависала.

И, в заключение осветим клавиши управления:

"W","S" - вниз, вверх;

"R","E" - влево, вправо;

"A","Z" - обороты двигателя;

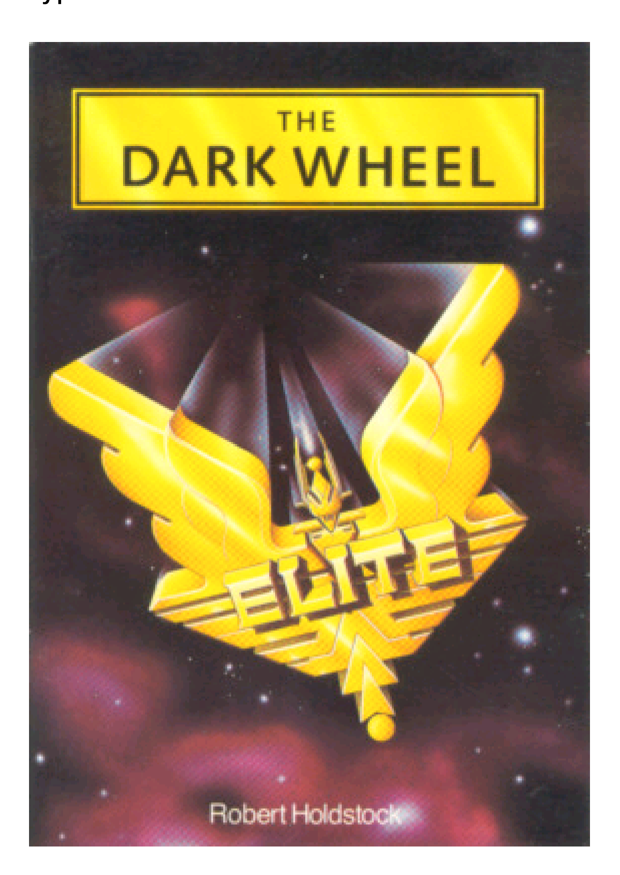
"T" - "огонь"

"CAPS SHIFT" - переключение оружия.

Счастливого полета!

THE DARK WHEEL

Для любителей программы "ELITE" мы продолжаем печать научно-фантастической новеллы английского писателя Роберта Хольдстока "The Dark Wheel" (перевод наш - "ИНФОРКОМ"). По расчетам новелла будет закончена в номере 5-6 "ZX-РЕВЮ" за 1993 год, а на очереди приключенческий боевик "Конференция", написанный по мотивам известного игрового сериала Колина Свинбурна JOE BLADE.



Продолжение. Начало см. на стр. 175, 176.

Наконец-то они вышли из гиперпространства.

В то же самое мгновение ловкие пальцы Джейсона взметнулись над клавиатурой управляющей консоли. "Авалония" рванулась вперед и одновременно начала закручиваться вокруг оси. Маленьким зеленоватым диском поплыла на экране планета Листи. Алекс видел, как отец навел и выпустил обе имевшихся на борту "Авалонии" ракеты и уже положил руку на триггер многоцелевого лазера.

Так значит это пират!

Когда до Алекса дошла в полной мере неотвратимость предстоящего боя, во рту пересохло, но мысли приобрели необычайную остроту и четкость. Никогда еще он не участвовал в боях, разве только на имитаторе. Конечно, отец рассказывал ему об этом, но в его рассказах это выглядело отнюдь не великолепным.

Итак, пиратский корабль, преследовал свою жертву на всем протяжении гиперперехода ради загруженного на борт фруктового сока!? Какой-то голос из глубины сознания подсказывал Алексу, что здесь что-то не так. Корсары так себя не ведут. Обычно они крейсируют на границах планетных систем, внимательно прощупывая пространство своими сканерами и придирчиво выбирая объект для атаки. Пираты могут повстречаться где угодно, но они редко оказываются в пределах корпоративных и демократических систем - здесь очень эффективно действует полиция. Их излюбленные системы - анархические и феодальные.

Определенно в поведении противника было что-то не то. Что-то не то, если это действительно пиратский корабль.

Алекс оторвал взгляд от проплывающей планеты и взглянул на сосредоточенное, посеревшее лицо отца. Судя по его выражению, их положение было далеко не безопасным.

- Одень дислок и займи спасательную капсулу, - буркнул Джейсон Райдер. - Выполняй!

- Я буду сражаться.

- Черта-с-два! Выполняй, я сказал! - с этими словами Джейсон сунул сыну в руки небольшую черную маску дистанционного локатора.

Защитные экраны "Авалонии" приняли первые удары ракет и пальцы Джейсона опять забегали по клавишам, приводя в действие защитные системы. Корабль дрожал от напряжения в последнем спасительном рывке. Начала работать система подавления вражеских ракет, а "Кобра" сделала второй залп.

Задний экран вспыхнул и залился ослепительным светом, затем сквозь яркое пятно вновь стали проступать серые очертания приближающегося корабля-убийцы.

Все произошло настолько быстро, что впоследствии Алекс много раз пытался восстановить точную последовательность событий и не мог. В ослепительной, но бесшумной схватке корабли вращались один относительно другого и оба вместе вокруг планеты. Пространство между ними полыхало. Оружие наносило удар за ударом, оружие отражало удары. А затем вселенная раскололась, огни "Авалонии" мигнули последний раз и погасли. Послышался свист уходящего воздуха, по управляющей консоли забегали огни индикаторов: перегрев лазера, низкий уровень энергии защитных полей, груз поврежден, температура в рубке падает…

Сильные руки подхватили Алекса, рванули вверх и грубо втолкнули в люк спасательной капсулы - вот все, что осталось в памяти о последних секундах жизни "Авалонии". Маска дислока в этот момент уже была на лице, закрывая глаза, нос и рот.

Корабль дрожал и скрежетал, топливо хлестало в космос. В последний раз отец и сын глядели в глаза друг другу.

-Я не понимаю, кому надо было…! - кричал Алекс сквозь грохот погибающего корабля.

- Ракксла. - успел сказать Джейсон. - Помни это, Алекс. Ракксла! Не забывай меня. Я не хотел тебе такой судьбы. Ракксла!

Сработала система катапультирования. Капсула закружилась в пространстве. В последний раз мелькнули гладкие обводы "Авалонии" и все застыло в ослепительной вспышке, а затем, сменяя друг друга, перед глазами пошли картины белого жара и черного холода.

В секунду ушло все, что было так близко и дорого - корабль, отец и частица прожитой жизни. Все распалось в огненной вспышке выстрела пиратского корабля. Языки пламени рванулись к спасательной капсуле. Жар, боль, холод - чувства менялись с калейдоскопической быстротой. Раздираемая на части, капсула была отброшена и, разваливаясь на куски, начала падение на планету. Последние капли жизни покидали бессознательное тело Алекса.

ГЛАВА 2.

Космос безмолвен, но крик о помощи услышит каждый.

Для того и служит спасательная маска дислока. В то самое мгновение, как капсула потеряла герметичность, струи пластифибры хлынули из сопел, заполняя кабину. Мгновенно твердеющая масса защитила безжизненное тело и от вакуума и от холода. Расход кислорода снизился до минимально необходимого для питания сердца и мозга. Включилась система поддержания жизнедеятельности, готовая сделать инъекцию адреналина или успокаивающего наркотика в случае необходимости. А дислок кричал о помощи на весь космос.

Это стандартное устройство выдавало мощный сигнал, мгновенно распознаваемый как крик о помощи безжизненного тела. Крик разносился на сорока каналах, четырежды в секунду меняя частоту на каждом из них. Сто двадцать шансов из ста за то, что он будет услышан.

Неуклюжий "Боа", загруженный от киля до рубки индустриальным оборудованием

замедлил ход и начал сканирование пространства в поисках источника тревожного сигнала.

Два полицейских "Вайпера" прервали дежурное патрулирование в окрестностях звезды и ринулись в установленный сектор.

Межзвездная пассажирская шхуна "Мори", переоборудованная в космический госпиталь, о чем свидетельствовала огромная золотая звезда на обшивке верхней палубы, начала медленно набирать ход.

Тысячи радиосообщений, непрерывно циркулирующие между кораблями, планетой и кольцом орбитальных станций были внезапно прерваны, уступив все эфирное пространство этому отчаянному крику о помощи. Повсюду прекратились телетрансляции. Изображения на экранах сменились картой близлежащих секторов с фиксацией координат аварийного дислока. Даже рекламные корабли прервали свою трансляцию, переключившись на поддержку поиска.

Тысячи людей устремили взгляд в звездное небо. Этот зов о помощи был слишком хорошо известен каждому и никто не мог оставаться спокойным.

Не прошло и двадцати секунд, как два автоматических спасательных зонда зависли над безжизненным телом попавшего в беду астронавта. Эти маленькие аппараты были, по-существу, автоматическими роботами и несли на себе запас кислорода на один час и до 40 доз различных медицинских препаратов, рассчитанных на оказание первой помощи. Когда спасательный трос притянул и зафиксировал тело, роботы приступили к работе. Проткнув защитный пластик, иглы вошли в тело и живительный кислород вместе с глюкозой и адреналином стал поступать непосредственно в кровь, а когда Алекс впервые открыл глаза, то сразу получил укол тенваля - сильного успокаивающего средства.

В ушах раздался голос говорящего робота.

- Брэнди? Скотч? Водка, сэр? Любые стимулянты на время ожидания.

- Что… случилось ... с кораблем?.. - с трудом выдыхал звуки сквозь маску Алекс.

- Так, значит брэнди - ответил робот и, ласково мигнув огоньками приборной панели, выдал больному двойную порцию квитирианского коньяку.

Через час Алекс уже был на борту космического госпиталя, дрейфующего над планетой. Медики позаботились об ожогах на руках и на лице, восстановили лопнувшие поверхностные кровеносные сосуды. Алекс чувствовал себя растоптанным и избитым, но физически был уже в норме.

Тем не менее, образ взрывающегося корабля продолжал преследовать его, как наваждение. Он стоял у широкого окна госпитальной палаты и сосредоточенно наблюдал за медленным вращением серо-зеленой планеты и суетливой толкотней челноков и грузовиков, снующих по своим делам в атмосферу и обратно. Разноцветные следы их пролета в верхних слоях атмосферы постепенно таяли и расплывались. Но куда бы Алекс ни обратил свой взор, везде ему казалось, что он видит "Кобру", ту самую "Кобру" и еще лицо отца.

Что-то было не так. Да, нападение было внезапным. Тревога, вспышка гнева, огонь и все кончено, но Алекс чувствовал, что все это время Джейсон Райдер ЗНАЛ.

Сейчас его сын ворошил онемевшую память и с каждой минутой все отчетливее сознавал, что отец гораздо лучше понимал нависшую над ними опасность, чем можно было судить по его виду. Теперь стало ясно, что это было в его лице, в напряженной атмосфере тревожного ожидания, нависшей в рубке, в отрывистых фразах, когда они еще только приближались к гипертуннелю.

Джейсон знал об угрозе и был готов к атаке. Он все подготовил для того, чтобы в последний момент спасти сына.

Это выглядело бессмысленно, но это было так. Теперь, когда Алекс потерял отца, у него не стало обоих родителей. У него вообще никого не осталось и родная планета сразу стала казаться чужим, негостеприимным миром.

За спиной мягко открылась дверь и вошла медсестра в сером халате. Слегка пожурив пациента за то, что тот покинул кровать, она по всей видимости осталась вполне удовлетворенной его состоянием. Затем поток посетителей стал непрерывным.

Сначала пришел доктор. Его, по-видимому, больше всего интересовало психическое

состояние больного и он, кажется, остался не вполне доволен.

- Молодой человек, - сказал врач, - вы потеряли отца и его уже не вернуть. Постарайтесь расслабиться. Нет ничего стыдного, если вы поплачете. Знаете, слезы смывают все - и горе и печаль. Не пытайтесь себя сдержать, это не пойдет вам на пользу.

- Я не плачу, я рыдаю о своем отце, и я еще сильнее буду рыдать, когда превращу в пепел того пирата, который его убил! И не раньше!

- Даже так?

- Именно так.

Следующим пришел агент Всегалактической службы медицинского страхования. Он взял необходимые данные и вскоре убедился, что Алекс застрахован на все случаи жизни, включая расходы на лечение и доставку на родную планету.

Затем пришла полиция. Два мужчины в серых плащах с серебряными поясами и с одинаковыми ничего не выражающими лицами представляли Департамент Борьбы с Распространением Наркотиков.

- Какой груз несла "Авалония"? Почему пират преследовал ее в пределах Корпоративной системы? Не занимался ли отец когда-нибудь перевозкой наркотиков? А оружия? Не возил ли он рабов? Какие инопланетные товары были на корабле? Манихуаза? Марсианский вирт? А может быть фиргланды? Что сказал отец перед смертью? Сможет ли Алекс узнать пиратский корабль? Отличительные приметы?

Алекс рассказал им все, что смог вспомнить, все, что видел, все, что слышал ... кроме того, что отец, очевидно, знал о готовящемся нападении, и ни слова не сказал о Ракксле.

Наконец полицейские ушли. Они совершенно не были удовлетворены тем, что смогли узнать.

- Молодой человек, мы понимаем, что Вы вполне самостоятельный пилот, имеете лицензию и можете сами выбрать путь домой, но мы настоятельно просим Вас перед отправлением согласовать с нами маршрут.

Ракксла!

Алекс наблюдал, как юркий, зловещий "Вайпер" отошел от госпиталя, круто развернулся и резко набрал скорость. Его серый цвет как нельзя лучше соответствовал мрачным штормовым облакам, затягивающим океан, над которым они пролетали.

Ракксла!

"Что это может быть? Что это может значить?"

После полуночи, когда Алекс еще не спал, в комнате замигал маленький зеленый огонек. Алекс зажмурился и понял, что за ним наблюдают.

- В чем дело? - обратился он в пустоту комнаты.

- По халофаксу для вас пришло сообщение. Просят дать ответный луч. Вы будете выходить на связь? - ответил голос медсестры.

Алекс присел. Дела развивались все любопытнее. Никто не мог знать, где он находится. Это факт. Он вновь нахмурился и ответил: - Да, конечно.

- Счет за переговоры записать на ваш кредит?

Опять загвоздка. Не было у него никакого кредита. У него вообще ничего не было, пока он не получит какую-то страховку. Ну, а раз так, то Алекс со спокойной душой ответил: - Да.

В центре комнаты воздух завибрировал, стал белым непрозрачным, потом рассыпался на мириады белых кристаллов, из которых постепенно стала собираться трехмерная голограмма мужской фигуры. Он был высоким, но слегка сутулился. Постепенно изображение приобрело цвет, но белизна фигуры сохранилась. Длинные седые волосы, всклокоченная борода, смуглое лицо. Маленькие блестящие глаза скрывались за глубокими морщинами. Человек улыбался. На нем была одета застиранная униформа торгового флота, одна рука безжизненно свисала вдоль тела. Даже ботинки были изрядно поношены и кое-где начала отставать подошва. Ручной лазер на боку тоже знавал лучшие дни, как и все в облике этого незнакомца.

- Это ты парень Райдера? - спросило поношенное изображение хриплым голосом человека, который подышал на своем веку глубоким вакуумом.

- Да, я Алекс Райдер. А вы?

Алекс слез с постели и подошел к фигуре. Старик спокойно смотрел на него и что-то жевал. Затем сплюнул. Плевок вроде бы пролетал мимо плеча Алекса и тот сделал шаг в сторону, забыв, что это только трехмерное изображение.

- Ты меня не помнишь, - сказал старик. - Да это и понятно. А я вот тебя помню.

- Как вас зовут?

- Рейф Зеттер. В прошлом торговец. Мы много лет работали в паре с твоим отцом, пока не разделили компанию, разойдясь во мнении по одному щепетильному вопросу.

- Рабы? - быстро отреагировал Алекс. Теперь и он вспомнил Рейфа. Но что же с ним стало! Он преждевременно состарился. Ему ведь столько же лет, сколько было и Джейсону Райдеру, но выглядел он лет на двадцать старше.

- Верно, парень. Рабы. Я прожил жизнь на конце вайперской удавки и был на пол шага впереди закона. К тому времени, как я позволил себе эту прихоть, у меня уже была железная задница и я сумел проскочить преисподнюю. Таково мое положение.

- В преисподней?

- Нет, я разорен.

Алекс кивнул, постепенно до него начинал доходить жаргон космических торговцев. "Железная задница" означает вооруженный до зубов корабль: силовые поля, ракеты, боевые лазеры. Такой может сделать пробежку по любой системе, даже по анархическому раю вроде Сотикью. "Проскочить преисподнюю" означает сорвать солидный куш на нелегальных операциях, хорошо погулять, а потом потерять все. Так они обычно и кончают.

Рейф продолжал.

- Мне очень жаль, что это произошло с Джейсоном. Он был хороший человек. Старый верный друг. Человек, которого я всегда буду уважать.

- Но это произошло не более, чем восемь часов назад. Откуда, черт возьми, вы можете об этом знать?!

Рейф закашлялся и сплюнул. Алекс не удержался и опять попытался уклониться от плевка, который растаял на границе голограммы. Холодок отвращения прошел по спине.

- Парень, твой темперамент такой же, как был у Джейсона. Не знаю, может быть даже ты унаследовал часть его способностей?

- Ответь на мой вопрос, старик. Как ты умудрился узнать об отце. Как ты меня нашел? - повысил тон Алекс.

Наблюдая из голограммы, Рейф пожевал и улыбнулся. Алекс невольно напрягся в ожидании очередного гиперпространственного плевка.

- Я повторяю, Алекс! Я с глубочайшим уважением отношусь к Джейсону. За то, кем он был и за все, что он сделал.

- Да, он был честный человек, - сказал Алекс, - и он был честным торговцем.

- Нет, он был много больше, чем ты думаешь, чертовски больше, - громко воскликнул РейФ и сплюнул. Алекс вздрогнул, завибрировало и стало тускнеть изображение.

- Что это значит?

Рейф Зеттер подался вперед, приблизившись к Алексу.

- Он был бойцом, Алекс, Одним из лучших. Он не должен был умереть так…

- Отец был торговцем, а не бойцом, - испуганно возразил Алекс.

- Подумай лучше, сынок.

- Но он не выносил стрельбы!

- Может быть, может быть, но это не останавливало его. А как ты думаешь он смог столько лет заниматься торговлей? Черт побери, Алекс, да ты можешь возить сметану и пряники и все равно рано или поздно найдется кто-то, кто захочет отнять их у тебя. Твой отец был бойцом высочайшего класса…

Алекс проглотил подступивший к горлу комок.

- Высочайшего класса?..

Рейф кивнул.

- Да, Алекс. - мягко продолжал он. - Ты можешь быть смертоносным, ты можешь быть опасным, но ты все равно превратишься в собачьи консервы на орбите какой-нибудь

собачьей планеты вроде Извивы. Но если ты Элита и если ты погибаешь, то значит для твоей смерти есть веские причины.

"Что говорит этот старик? Элита? Боец класса Элита?" У Алекса закружилась голова. Он слышал о пилотах, которые дослужились до этого рейтинга - их было очень немного. Очень многие были Опасными, иначе и нельзя заниматься торговлей. Очень многие были Смертоносными. Их много как среди торговцев, так и среди пиратов. Но Элита?! Таких единицы.

Его отец, Джейсон Райдер был Элитой, а никто в семье даже и не догадывался!

- Джейсон был одним из самых лучших. Ты, вероятно, никогда не видел его корабль. Это просто крепость. Он торговал в таких местах, которые мы видели разве что в кошмарах. - Рейф восхищенно покачал головой. - Один из лучших… Боец высочайшего калибра… - Его взгляд вновь упал на Алекса. - Весь вопрос в том, сможешь ли ты стать таким же?

- Почему ты сомневаешься?

- Джейсон никогда не рассказывал о тебе. Я думаю, он берег тебя. Беда в том, что мне не с чего теперь начать. По твоим глазам я вижу, что ты будешь мстить за отца, но для меня это означает лишь то, что еще один Райдер станет космической пылью еще до того, как сумеет навести ракеты.

Алексу не понравился этот тон.

- Я провел многие часы на тренажерах и у меня высшие баллы.

Рейф рассмеялся и смачно сплюнул, а затем сказал серьезно.

- Алекс, хотел бы я знать, не собираешься ли ты …

- … превратиться в собачьи консервы на орбите Извивы?

- Да, что-то в этом роде. Единственный человек, который знал, на что ты годишься, был твой отец. А теперь. Алекс, ответь мне. Скажи мне правду… Отец тебе ничего не сказал в тот момент… ну, в общем, что он сказал тебе перед смертью? Может быть, он на что-то намекнул?

- Он много чего говорил, - промямлил Алекс и почувствовал горькую боль, вспомнив глаза отца и его последние слова: "Не забывай меня. Алекс..." - Я думаю, он знал, что погибнет, последнее слово, которое он произнес, было "Ракксла". Я не знаю, что это может означать. Думаю, что что-то инопланетное.

Рейф улыбнулся и покачал головой, в его глазах вспыхнул блеск.

- Ракксла - это не что-то инопланетное. Это призрачный мир. Планета - легенда. - Он еще немного поколебался и продолжил: - отец на самом деле это сказал?

Алекс кивнул: - За мгновение до... Он сказал это перед смертью, это было его последнее слово.

- Значит, он знал и меня это радует. Тогда так. Завтра, Алекс, ты вылетаешь на Тионислу. Там возьмешь орбитальный челнок и полетишь на кладбище погибших кораблей. Скажешь, что прибыл посетить могилу звездопроходца Флейшера и внимательно смотри по сторонам. Сделай это, парень. Завтра я буду тебя ждать.

- Ждать ради чего?

Рейф закашлялся. - А как ты собираешься охотиться на "Кобру? Ты будешь летать по космосу на попутках? А воевать будешь чем? Размахивая дубиной? Тебе нужен корабль. Будь на свалке в Тионисле. Я знаю один корабль, который тебе нужен. И никому ни слова. Молча отправляйся завтра на Тионислу.

- Но…

- Прощай, Алекс!

И Рейф сплюнул в последний раз.

(Продолжение следует)



"ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

Дорогие друзья!

Мы заканчиваем печать ZX-РЕВЮ образца 1992 года. Самое время подвести некоторые итоги, обсудить возникшие проблемы и наметить планы на будущее.

Основным итогом, конечно, является то, что ZX-РЕВЮ благополучно пережило второй год своего существования и, несмотря на то, что эти последние номера Вы получаете с очевидным опозданием, мы с оптимизмом смотрим в будущее и полномасштабный выход в свет ZX-РЕВЮ-93 сейчас уже не вызывает сомнений.

К отрицательным итогам нам придется отнести тот печальный факт, что многократно повышены почтовые тарифы между Россией и другими странами СНГ. Это практически лишает многих наших постоянных читателей из этих стран возможности получать ZX-РЕВЮ и другие материалы.

Мы, наверное, никогда не поймем, почему почтовые отправления на Украину и в Беларусь должны оплачиваться в несколько раз дороже, чем в Россию, а наших постоянных читателей из республик Прибалтики вообще приравняли по тарифам к жителям Мадагаскара.

Мы надеемся, что когда-нибудь нормально мыслящие политики еще скажут свое слово о роли Министерств Связи стран СНГ в деле укрепления гуманитарных связей и воздадут им по заслугам, а пока будем как-то выкручиваться и искать обходные пути.

Итак, начнем по порядку.

1. Каким будет "ZX-РЕВЮ" в 1993 году.

В основном мы оставим все без изменений. Судя по вашим письмам, содержание и структуру ZX-РЕВЮ, достигнутые в 1992 году можно считать более удачными, чем в 1991г., и ничего радикально мы менять не будем.

Единственно, чего будет больше - это раздела "Маленькие хитрости". Причем, мы уже сегодня можем сказать, что основной темой этого раздела будут разного рода манипуляции и операции с экраном (как правило, в машинных кодах).

Мы добьемся некоторого баланса, когда интересные приемы, приводимые в этом разделе, будут служить как бы дополнительными иллюстрациями к тем книгам, которые мы готовим и выпускаем по графике "Спектрума". Хотя это вовсе не означает, что для понимания и применения предлагаемых материалов необходимо эти книги иметь. В то же время, для тех, кто хочет разобраться с этими вопросами досконально, они лишними не будут.

Так же, как и в 1992 году, мы будем печатать и крупные материалы с продолжением. Одним из них станет сериал Дэвида Новотника, посвященный экспертным системам вообще и возможностям их реализации на "Спектруме" (перевод и редактирование - наше).

Вторым крупным материалом станет книга Стюарта Николса, посвященная вопросам создания игровых программ на АССЕМБЛЕРе. Перевод подготовлен нашим соавтором из г. Балашова Саратовской области - В.Паворинын. Книга заинтересует тех, кто еще не освоил машинный код, поскольку в ее методическую основу положен принцип постепенного "перевода" команд и операторов БЕЙСИКа в их машинно-кодовые аналоги. Книга очень

доступна для понимания и потому мы ее смело предлагаем массовому читателю, а не выпускаем отдельным выпуском для избранных.

В ближайшие месяцы мы закончим публикацию материалов, посвященных вопросам постановки и снятия защиты программ, но на очереди еще интересные статьи, содержащие разбор методов работы известных "хаккеров".

Готовится крупный материал о возможности применения "Спектрума" в астрологии. Только не пугайтесь, никакой мистики не будет. Речь пойдет о применении компьютера к анализу движений небесных тел. Кстати, первые эксперименты показали, что по производительности "Спектрум" почти не уступает в этой области IBM PC XT.

Мы по-прежнему будем уделять часть места исследованию игровых программ и будем экспериментировать с оригинальными литературно-художественными интерпретациями игровых программ. Основная цель этих экспериментов - выйти на новый жанр "компьютерной новеллы", которая должна, с одной стороны, давать пользователю полезную информацию о технике прохождения той или иной игры, а с другой стороны, делать это так, чтобы не отбивать, а наоборот пробуждать у него желание сыграть в эту игру.

Без изменения мы сохраним остальные рубрики "ZX-РЕВЮ".

Как обычно, в каждом номере будут статьи, посвященные применению "Спектрума" в школе. При наличии интересных материалов, поступающих от читателей, будет действовать раздел "Форум" и, конечно, для опытных программистов, которым есть чем поделиться, открыт раздел "Профессиональный подход".

2. Как можно будет подписаться на "РЕВЮ-93".

Большой разнобой в условиях поставки в разные республики СНГ не позволяет нам однозначно всем сообщить условия подписки. Пока можем сообщить только, что принято принципиальное решение о том, что прием подписки по почте мы все-таки проведем. Во-первых, непропорционально выросли цены на железнодорожные и авиабилеты, что не позволяет многим читателям посетить корпункт. Кроме того, не прекращается поток писем с просьбой о приеме подписки по почте в порядке исключения. Честно говоря, видя желание многих наших подписчиков получать "ZX-РЕВЮ" и в 1993 году, мы не имеем возможности им отказать.

С другой стороны, у нас есть определенные трудности в приеме подписки по почте, поэтому эти условия мы оговорим в прилагаемом к данному выпуску информационном листке, т.к. он попадает ТОЛЬКО К НАШИМ ЗАРЕГИСТРИРОВАННЫМ ЧИТАТЕЛЯМ и только они смогут им воспользоваться.

Наш постоянный корпункт продолжает действовать по адресу: г. Москва, ул. Новый Арбат (бывший просп. Калинина), д. 2, отделение связи Г-19. На первом этаже в операционном зале Вы найдете нас по вывеске "ZX-РЕВЮ". Корпункт работает все дни недели, кроме воскресенья с 10 до 17 часов (перерыв на обед с 14 до 15 часов).

Несмотря на то, что мы приняли решение о приеме подписки по почте, у Вас есть по крайней мере четыре причины, по который посещение корпункта было бы желательным:

- цены на корпункте на всю имеющуюся у нас литературу, включая и подписку на ZX-РЕВЮ-93, существенно ниже, чем при оплате по почте;

- при покупке мелкооптовых партий (от 10 экз.) действуют дополнительные скидки;

- только здесь можно приобрести некоторые материалы прошлых лет (описания языков программирования, различных прикладных программ и т.п.), которые у нас заканчиваются и потому из свободной продажи по почте изъяты;

- через корпункт мы начали распродажу остатков кассет с программными сборниками прошлых лет. Обращаем Ваше внимание на то, что по причине их малого количества, по почте они уже давно не высылаются, и заказы на них мы не принимаем.

Вам совсем не обязательно являться на корпункт лично. Вы можете попросить об этом кого-либо из своих друзей и знакомых, находящихся в Москве проездом.

Вторая возможность контакта с нами через радиорынки. На сегодняшний день в

Москве функционирует крупный радиорынок в Тушино (ст. метро Тушинская). По субботам и воскресеньям (с 9 до 13) там можно войти в контакт с нашими дистрибьюторами для решения оперативных вопросов. Местные органы власти планируют закрытие данного радиорынка. В этом случае он может быть перенесен в другие точки, где также можно будет найти наших дистрибьюторов. Есть предварительные сведения о планах переноса Тушинского радиорынка в район платформы "Трикотажная" (проезд на электричке от Рижского вокзала или от платформы "Тушино").

И, наконец, мы решаем вопрос об открытии представительств в республиках СНГ. В настоящий момент ведутся переговоры с украинской фирмой из г. Днепропетровска, которая возьмет на себя обслуживание жителей Украины нашими материалами. При положительном решении данного вопроса фирма-представитель войдет с Вами в контакт, предложит свои услуги и начнет обслуживание как тех, кто подписался у нас, так и своих подписчиков.

Надеемся, что и в других странах СНГ найдутся организации, имеющие желание и возможность стать нашими официальными представителями. Это позволят не только сохранить число наших читателей в этих странах, но и значительно его увеличить. Мы готовы к проведению подобных переговоров.

3. Наши новые разработки.

3.1. Выпущен годовой комплект "ZX-РЕВЮ-91" в виде одной аккуратной книжки размером 200 х 140 мм, 254 стр.

3.2. К тому времени, как вы прочтете эти строки, будет выпущена аналогичная книжка, содержащая полный комплект "ZX-РЕВЮ-92".

3.3. Вышло новое издание книги по программированию в машинных кодах. В новое издание вошли все три тома первого издания ("Первые шаги в машинном коде", "Практикум по программированию в машинных кодах" и "Справочник по программированию в машинных кодах".).

В отличие от первого издания, новое издание существенно расширено (более чем на 20%). Дополнение коснулось следующих тем:

- применение прерываний второго рода;
- концепция каналов и потоков;
- вопросы русификации компьютера;
- описание директив АССЕМБЛЕРа.

Общий объем новой книги – 272 стр.

3.4. На днях выходит из печати первый том четырехтомника, посвященного графике "Спектрума". Этот том называется "Элементарная графика" и содержит 208 стр. Готовится к передаче в печать второй том - "Прикладная графика".

4. Вопросы лицензирования.

Ранее мы объявляли нашим читателям о готовности передачи компьютерного текста наших книг для издания тиража в регионах на условиях ограниченной лицензии (см. стр. 133-134).

Сейчас мы можем сообщить, что такая основа для сотрудничества вызвала живейший интерес всей страны. Вместе с тем, вскрылись некоторые проблемы.

Так, например, все без исключения участники переговоров о покупке лицензии на право издания и распространения наших книг в своих регионах отметили низкий процент лицензионных отчислений, который мы получаем и который их полностью устраивает.

Основной проблемой оказались цены на услуги печати на местах. Так, например, в некоторых районах России и Украины только печать книжки объемом 200-220 стр. требует до 300-500 рублей за экземпляр. Многим покупать готовую продукцию у нас оказалось дешевле, чем печатать у себя. В итоге нами проданы пока три лицензии на печать книги "Элементарная графика" в следующие регионы: Чувашия, Нижний Новгород, Сахалинская

обл. Переговоры с еще 10-15 партнерами пока не закончены.

Мы очень рекомендуем тем, кто захочет получить от нас лицензию на издание и распространение наших книг стараться не пользоваться услугами крупных специализированных типографий, а сосредоточиться на небольших ведомственных типографиях или, еще лучше, принадлежащих предприятиям. С их помощью можно сделать печать более рентабельной.

Вместе с тем, мы продолжаем расширять перечень книг, доступных для издания по регионам. Сегодня мы предлагаем желающим получить на дискете и издать у себя нашу книгу, посвященную программированию в машинных кодах (расширенный и дополненный трехтомник).

В течение ближайшего времени будет предложен и второй том графики - "Прикладная графика".

Для проведения консультации и переговоров по вопросам покупки лицензий, открытия представительств в странах СНГ, а также для желающих приобрести литературу в среднеоптовых количествах (от 100 экз.) в прилагаемом информационном листке мы даем контактный телефон.

5. Новые разработки для IBM-совместимых компьютеров.

В ближайший месяц мы оповестим наших дилеров о новых разработках обучающих и деловых программ для IBM совместимых машин. Подготовлены программы для детей по арифметике, математике и информатике, а также выпущены две версии универсальной и очень удобной системы для создания баз данных - "DB-процессор", который может стать прекрасной основой для Вашего частного предпринимательства.

А сейчас мы прощаемся с Вами. До свидания и до новых встреч!

"ИНФОРКОМ"

BETA BASIC

Окончание.
(Начало см. на стр. 3, 47, 91, 135, 179).

6. DPEEK (адрес).

FN P(адрес).

См. также DPOKE.

Функция DPEEK - это то же самое, что и двойной PEEK. Эквивалентом этой функции в стандартном БЕЙСИКе является выражение:

```
LET a = PEEK (addr) + 256*PEEK (addr+1)
```

Таким образом, эта функция выдает содержимое двух следующих друг за другом байтов. Обратите внимание на то, что младший байт идет первым. Это очень удобно для проверки содержимого системных переменных и при анализе машиннокодовых процедур. Например:

```
10 LET nxt = DPEEK(23637):POKE nxt+5.65
20 REM XXXXX
```

В десятой строке будет прочитано содержимое системной переменной NXTLN, после чего в результате POKE первый символ, следующий после оператора REM будет изменен на символ "A". Сдвиг на +5 байтов в строке 20 необходим для того, чтобы пропустить номер строки (2 байта), длину этой строки (2 байта) и код самого оператора REM (1 байт).

Оператор DPOKE обеспечивает двойной POKE точно так же, как функция DPEEK обеспечивает двойной PEEK.

7. EOF (номер потока).

FN E(номер потока).

Название функции EOF происходит из сокращения End of File (конец файла). При работе с микродрайвом эта функция говорит о том прочитан или нет последний байт из загружаемого файла.

Поток, номер которого указан в функции, должен быть предварительно открыт (OPEN#) для данного файла, содержащегося на ленте микродрайва. Если этого не сделать, то Вы получите сообщение об ошибке "Invalid Stream". Этот файл должен, конечно, физически существовать и быть открытым для чтения, иначе вы получите сообщение об оошибке "Reading a "Write" file"

Функция выдает "1", если последний байт данных считан из файла или "0", если он еще не считан. Это помогает точно установить момент конца считывания данных и избежать попыток прочтения большего количества данных, чем в этом файле есть, что вызвало бы появление ошибки.

Предположим, что на микродрайве у вас есть файл "data". Наиболее элегантный путь использования функции EOF выглядел бы так:

```
10 OPEN #5,"m",1,"data"
20 DO UNTIL EOF(5)=1
30 INPUT #5;a$: PRINT a$
40 LOOP
50 STOP
```

Впрочем, строку 20 можно записать еще короче:

```
20 DO UNTIL EOF(5)
```

Если Вы еще не привыкли к использованию циклов DO...LOOP, то можете попробовать действовать так:

```
10 OPEN #5,"m",1,"data"
20 INPUT #5;a$: PRINT a$
30 IF EOF(5)=0 THEN GO TO 20
40 STOP
```

8. FILLED ().

FN F().

См. также команду FILL.

Эта функция дает количество пикселов, включенных по последней команде FILL. Например:

```
10 PLOT 0,0: DRAW 9,0: DRAW 0,9
20 DRAW -9,0: DRAW 0,-9
30 FILL 5,5
40 PRINT FILLED()
```

Длина стороны квадрата - 10 пикселов. (Мы давали число 9 в команде DRAW, но поскольку реальная линия на экране имеет толщину не менее одного пиксела, то и сторона квадрата у нас будет десять пикселов.) Внутренняя неокрашенная часть квадрата будет иметь линейный размер по 8 пикселов на сторону, и функция FILLED() даст нам значение 64.

Если Вы попробуете в строке 30 дать команду на стирание нарисованного квадрата:

```
30 FILL PAPER; 5,5
```

то функция FILLED() даст Вам результат - 100.

Разница между 100 и 64 - количество пикселов, ушедших на изображение периметра квадрата.

9. HEX$ (число).

FN H(число).

См. также функцию DEC(строка).

Эта функция конвертирует десятиричный числовой аргумент в шестнадцатиричную символьную строку. Строка имеет два символа, если число было в диапазоне от -255 до +255 или четыре символа, если абсолютная величина числа была больше. Если же число по абсолютной величине больше, чем 65535, то выдается сообщение об ошибке "Integer out of range".

```
HEX$ (32)       = "20"
HEX$ (255)      = "FF"
HEX$ (512)      = "200"
HEX$ (-64)      = "C0"
HEX$ (-1024)    = "FC00"
```

Возможность работы с отрицательными целыми числами будет полезна тем пользователям, которые работают с машинным кодом. В частности, это пригодится при расчете адресов обратных относительных переходов.

Эта функция очень удобна, если Вам надо распечатать содержимое памяти в шестнадцатиричном виде:

```
10 INPUT "Start address? "; addr
20 PRINT HEX$ (addr); " "; HEX$(PEEK addr)
30 LET addr=addr+1: GO TO 10
```

Если же Вы хотите и начальный адрес при вводе тоже задавать в шестнадцатиричном виде, то можете переделать строку 10 например так:

```
10 INPUT "start address? "; A$: LET addr = DEC (A$)
```

10. INARRAY (строковый массив (начальный элемент)<,границы>,искомая строка).

FN U (строковый массив (начальный элемент)<,границы>,искомая строка).

См. также функцию INSTRING.

Функция INARRAY осуществляет сканирование строкового массива в поисках заданной строки. Если она не найдена, то выдается 0. Если же она разыскана, то выдается номер той строки в массиве, в которой данная символьная последовательность встретилась впервые.

В принципе функция INARRAY - разновидность функции INSTRING, но предназначенная для работы с массивами, поэтому будет неплохо, если Вы сначала прочитаете про работу функции INSTRING.

Нижеприведенный пример показывает, как можно разыскать все символьные сочетания "howdy" в заданном массиве

```
10 DIM a$(20,10)
20 LET a$(RND*19+1) = "howdy"
30 LET num= 1
40 DO
50   LET num=INARRAY (a$ (num),"howdy")
60 EXIT IF num=0
70   PRINT num;" ";a$(num)
80   LET num=num+1
90 LOOP UNTIL num > 20
```

В строке 50 выражение a$ (num) при num=1 предполагает, что поиск начнется с первой строки массива. Когда первое искомое сочетание символов будет найдено, строка, в которой оно содержится, будет выдана на печать, после чего поиск будет продолжен с (n+1) строки.

Оператор EXIT IF выведет программу из цикла, когда уже не останется неразысканных сочетаний "howdy". Оператор LOOP UNTIL в строке 90 безусловно прекратит работу программы, когда все элементы исходного массива будут просмотрены.

Программа проверит каждую строку полностью, но Вы можете ограничить пределы поиска внутри каждой строки исходного массива, задав границы поиска.

```
50 LET num=INARRAY(a$(num,3 TO 7),"howdy")
```

Эта возможность позволяет Вам разрабатывать и манипулировать с базами данных. Например, вам надо найти в базе все адреса, относящиеся к городу Брест. Тогда вы проведете поиск именно по тем участкам информации, которые имеют отношение к полю "Город", и исключите тем самым жителей города Москвы, проживающих на Брестской улице, поскольку поле "Улица" в розыск не попадет.

При поиске вы можете использовать символ-заместитель "#", который обозначает "любой символ". Это же, кстати, относится и к функции INSTRING. Единственный случай, когда этот символ ничего не замещает, а принимается сам за себя, - это когда он стоит первым символом в искомой символьной строке.

Вы можете практиковать довольно сложные условия для поиска. Например, у Вас в базе данных содержится список лиц с именами, фамилиями, адресами, цветом волос и т.п. Если каждое поле начинается строго с фиксированной позиции, а так и должно и быть в структурированной базе данных, то Вы можете найти всех содержащихся в ней жителей Лондона, имеющих коричневый цвет волос. Для тех позиций, которые соответствуют городу, Вы введете London, для тех, которые соответствуют цвету волос - Brown, а для всех прочих поставите символы "#".

Нижеприведенный пример покажет как это можно сделать. Здесь первые 20 символов отведены под имя персоны, а последующие 15 - под название города.

```
10 LET namelen=20, townlen=15
20 DIM d$(10,namelen+townlen)
30 FOR n=1 TO 10
40   INPUT "name? ";n$
50   INPUT "town? ";t$
60 LET d$(n,1 TO namelen)=n$
70 LET d$(n,namelen+1 TO )=t$
80 NEXT n
90 PRINT "массив заполнен"
```

Мы подготовили исходный массив для дальнейшей работы. Следующий программный блок позволит Вам найти заданную комбинацию имени и адреса. Примененная здесь функция STRING$ служит для генерации заданного количества символов "#", служащих для отделения между собой полей имен и адресов.

```
100 INPUT "name=? ";n$
110 INPUT "town=? "; t$
120 LET s$=n$ + STRING$ (namelen-LEN n$,"#")+t$
130 LET loc=INARRAY(a$(1),s$)
140 IF loc=0 THEN
        PRINT "Not found "
    ELSE
        PRINT loc;" ";d$(loc)
150 GO TO 100
```

Примечание: функция INARRAY не может работать с массивами, размерность которых больше, чем 2.

11. INSTRING (старт, строка 1, строка 2).

FN I (старт, строка1, строка2).

См. также функции: MEMORY$, INARRAY.

Функция INSTRING просматривает строку 1 в поисках строки 2, начиная с символа "старт". Если такое вложение найдено, то функция выдает порядковый номер символа в строке 1, с которого начинается строка 2, в противном случае выдается ноль.

Первая строка может быть любой длины, а вторая - не более 256 символов, иначе будет выдано собщение об ошибке "Invalid argument". Если начальная позиция для поиска "старт" равна нулю, будет выдано сообщение об ошибке "Subscript wrong".

В тех случаях, когда длина второй строки больше, чем первой, а также когда "старт" больше, чем длина первой строки, функция выдает ноль.

В искомой строке можно использовать символ-заместитель "#", который служит вместо тех символов, которые при поиске не имеют значения. Например:

```
PRINT INSTRING (1,A$,"SM#TH")
```

найдет появление SMITH, SMYTH, SMATH и т.п. в символьной строке A$.

Единственный случай, когда символ "#" воспринимается буквально, т.е. служит вместо самого себя - когда он стоит первым символом в искомой строке.

Возможность задания начальной позиции для поиска будет полезной в тех случаях, когда Вы рассчитываете найти не одно, а большее количество вхождении искомой строки в исходную. Ниже приведен пример, который разыщет строку "TEST" в символьной строке A$.

```
100 DIM a$(1000)
110 FOR n=1 TO RND*10 + 3
130 LET pos = RND*995
130 LET A$ (pos TO pos + 3)
140 NEXT n
150 PRINT "Press any key"
160 PAUSE 0
170 LET loc=1
180 LET loc = INSTRING (loc,a$,"TEST")
190 IF loc <>0 THEN PRINT "Found"; loc: LET loc=loc+1:GO TO 180
200 PRINT "Finish"
```

Строка A$ начинает просматриваться с позиции 1 (loc=1), пока не будет найдена последовательность символов "TEST". После этого поиск будет продолжен со следующего символа (loc=loc+1). Когда функция выдаст 0, поиск будет завершен.

Обратите внимание на то, что строку 190 можно немного упростить:

```
190 IF loc THEN PRINT ...
```

Если Вам надо сделать поиск по оперативной памяти или по ее части, Вы можете воспользоваться функцией MEMORY$.

Функцию INSTRING можно с успехом использовать в обучающих и в некоторых игровых программах (в частности в адвентюрных). Предположим, что программа задала пользователю вопрос, правильный ответ на который хранится в переменной c$="NAPOLEON". Те, кто введут в качестве ответа "NAPOLEON "(обратите внимание на финальный пробел) или "NAPOLEON BONAPARTE", будут разочарованы, поскольку программа воспримет эти ответы, как неправильные. А это очень часто случается, если подпрограмма сравнения ответа и эталона не обладает достаточной гибкостью. Функция INSTRING поможет справиться с этой проблемой. Она оценит как правильный любой ответ, при котором контрольная строка входит в состав введенной пользователем.

```
INPUT a$: IF INSTRING (1,a$,c$)<> 0 THEN PRINT "Correct"
```

Вы можете решить при этом и проблему регистра, если заранее не знаете большими или малыми буквами будет набран ответ пользователя. Вам надо принудительно конвертировать его ответ в прописные буквы с помощью функции SHIFT$:

```
LET a$=SHIFT$(1,a$)
```

Еще одно возможное применение функции INSTRING - для упаковки нескольких строк в одну длинную строку. При этом Вы можете, например использовать символы с 1-го по 31-

ый в качестве "маркеров" входящих строк.

Так, символ CHR$ 1 будет отмечать начало первой подстроки в генеральной строке, символ CHR$ 2 - начало второй подстроки и так далее. Так хранить строки гораздо компактнее, чем в массиве, если они имеют неодинаковую длину. Найти строку с номером n в генеральной строке будет несложно:

```
PRINT a$(INSTRING(1,a$,CHR$ n)+1 TO INSTRING (1,a$,CHR$(n+1)-1)
```

Возможны и многие другие интересные пути использования функции INSTRING.

12. ITEM ()

FN T()

См. также раздел, посвященный процедурам.

Эта функция дает информацию о следующей единице данных, которые подлежат вводу через READ. Как правило, функция используется при работе с процедурами, но может быть применена и при обычной технике READ...DATA. Функция возвращает следующие значения:

0 - все данные из текущего оператора DATA прочитаны. Текущий оператор DATA может быть при этом и списком параметров, следующим за вызовом процедуры.

1 - следующий объект - символьный.

2 - следующий объект - числовой.

При работе с процедурами функция ITEM() может дать информацию о физической природе первого объекта данных, но во всех остальные случаях она будет выдавать 0 до тех пор, пока хотя бы один объект не будет считан из списка DATA с помощью READ. В нижеприведенном примере проверяется ITEM() после READ.

```
100 DO
110    READ x
       PRINT x
120 LOOP UNTIL ITEM()=0
130 DATA 1,2,3,4,5,6
```

Таким образом, строки 100...120 могут считывать строки DATA произвольной длины.

13. LENGTH (n, "имя массива")

FN L(n,"имя массива")

Функция LEHGTH выдает размер массива. Для Бета-Бейсика это особенно важно, поскольку этот язык программирования позволяет во время работы проводить изменения длин массивов без потери данных. Эта же функция может быть использована для определения местоположения числовой или символьной последовательности в оперативной памяти компьютера.

Параметр n определяет о какой размерности для двумерных массивов идет речь. Если n=1, то функция возвращает размер массива в первом измерении, а если n=2, то во втором измерении (или единицу, если массив одномерный). С массивами размерности больше, чем 2, эта функция работать не может.

В имени массива значимыми являются только первые два символа, поэтому следующие имена будут приняты, как правильные: a$, b$, C(, d(), a$QWERT. Если вместо имени массива ввести имя простой символьной переменной, то она будет интерпретироваться как одномерный массив, состоящий из односимвольных элементов, количество элементов при этом равно длине символьной строки.

Примеры:

```
10 DIM a$(10,20)
20 PRINT LENGTH (1,"a$"): (10)
30 PRINT LENGTH (2,"a$"): (20)
40 DIM b(5)
50 PRINT LENGTH (1,"b("): 15)
60 PRINT LENGTH (2,"b("): (1)
```

функция имеет еще одно замечательное свойство. Если вместо параметра размерности массива ввести 0, то она выдаст адрес, в котором в памяти компьютера расположен первый элемент массива или символьной строки:

```
LENGTH (0, имя массива)
```

Эта возможность может быть использована теми, кто программирует в машинных кодах - можно найти адрес, в котором хранятся те или иные данные и даже можно найти адрес, с которого начинается процедура, записанная в машинных кодах, если она оформлена, как массив данных. Для тех же, кто программирует на БЕЙСИКе, имеется тоже немало интересных приложений, особенно если этой функцией пользоваться совместно с оператором POKE и функцией MEMORY$. Так, например, можно продублировать массив a$ в массиве b$, что может быть полезным при реорганизации данных:

```
20 LET e=LENGTH(1,"a$")
30 LET f=LENGTH(2,"a$")
40 DIM b$(e,f)
50 LET start = LENGTH(0,"a$")
60 POKE LENGTH (0,"b$"),MEMORY$()(start TO start+e*f-1)
```

Примечание: если Вы перебрасываете не символьный, а числовой массив, то в строке 60 следовало бы вместо e*f подставить 5*e*f, поскольку каждое действительное число в "Спектруме" хранится в пятибайтной форме.

Приведенный выше алгоритм во многом отличается от команды Бета-Бейсика COPY для массивов. Он, конечно, менее удобен, но может служить неплохой основой для создания на его базисе быстро работающей процедуры копирования массивов. Добавьте для этого строки:

```
10 DEF PROC dup REF a$, REF b$
70 END PROC
```

И теперь такая команда, как dup r$,t$ продублирует символьную строку r$ в строке t$.

Замечание для опытных пользователей:

Если у Вас уже есть ранее созданный массив данных, который Вы хотели бы использовать вместе с программой, написанной на Бета-Бейсике, но который слишком велик для загрузки, Вы можете разделить его на части и представить в виде блоков кодов (CODE), а затем использовать функцию LENGTH для того, чтобы программно перебрасывать эти блоки кодов в массив.

Без Бета-Бейсика Вы можете найти начало своего массива или символьной строки, если дадите им такое имя, которое обеспечит им первое место среди переменных в соответствующей области. Сделайте их объявление первым в программе. Затем найдите искомый адрес:

```
PRINT PEEK 23627+256*PEEK 23628+d
```

Здесь d=3, если Вы ищете символьную строку: d=6, если Вы ищете начало одномерного массива и d=8, если массив - двумерный.

Так Вы получите адрес первого байта данных в области переменных, поскольку символы, строки и числовые ряды в памяти идут друг за другом в линейной последовательности. Вы легко сможете рассчитать стартовый адрес и длину блока своего массива. Помните, что символы занимают по одному байту, а числа - по пять.

Примечание: поскольку с помощью оператора LET могут создаваться новые символьные строки или изменяться ранее существовавшие, то значения, ранее полученные функцией LENGTH (0,"имя") могут устаревать.

14. MEM ()

FN M()

Эта функция выдает количество свободных байтов в оперативной памяти компьютера. В скобках ставить ничего не надо. Попробуйте, например:

```
PRINT MEM(): DIM a$(100):PRINT MEM()
```

Эта простая функция содержит в себе в основной только вызов процедуры из области ПЗУ. Если Вы работаете в стандартном Бейсике, то выполнить ту же операцию можете командой

```
PRINT 65535 - USR 7963
```

15. MEMORY$ ()

FN M$()

См. также POKE симв. строка.

Эта функция выдает содержимое оперативной памяти компьютера в виде символьной строки. Правда, при этом содержимое нулевой ячейки памяти не включается. Поэтому команда

```
CODE MEMORY$()(1)
```

- то же самое, что и PEEK 1. Последние три байта оперативной памяти также по техническим причинам исключены, поэтому эта функция имеет максимальную длину 65532.

Конечно, если Вы попробуете что-то типа:

```
LET a$ = MEMORY$ (),
```

то Вам не хватит оперативной памяти, чтобы разместить там переменную a$, но пользоваться этой функцией надо по-другому. Например:

```
LET a$=MEMORY$()(16364 TO 22527)
```

Если учесть способность Бета-Бейсика выполнять POKE для символьных строк, то эта функция дает программисту возможность манипуляций с большими блоками памяти при очень высокой скорости (в отличие от стандартного БЕЙСИКа). Более подробно возможности применения этой функции Вы найдете в разделе, посвященном оператору POKE.

Другое применение MEMORY$ - для сканирования оперативной памяти при использовании INSTRING. Вы можете просканировать заданную секцию в памяти, указав пределы в функции MEMORY$ () (23759 TO ...). Но функция INSTRING работает настолько быстро, что этим можно и не пользоваться, а сканировать по всей памяти целиком.

```
10 REM asdfg
20 PRINT INSTRING (1,MEMORY$(), "asdfg")
```

Эта программа найдет строку asdfg в операторе REM в строке 10. Если Вы опустите строку 10, то asdfg будет найдена в строке 20. Если Вы присвоите:

```
LET a$ = "asdfg",
```

то эта символьная строка будет найдена в области программных переменных. Таким образом, где-нибудь в памяти, но Вы найдете содержимое любой символьной строки.

Вместо "1" в функции INSTRING мы могли бы дать DPEEK (23635) , т.е. содержимое системной переменной PROG для того, чтобы начинать поиск не от начала ПЗУ, а с того места, откуда начинается БЕЙСИК-программа.

Если же Вы хотите найти все случаи повторения в памяти заданной последовательности символов (байтов), то можете делать так:

```
10 LET adr=1
20 LET adr=INSTRING(adr, MEMORY$ (), a$)
30 IF adr THEN PRINT adr: LET adr=adr+1: GO TO 20
```

Поскольку Бета-Бейсик позволяет выполнение операции POKE для символьных последовательностей, Вы можете легко организовать поиск нужной последовательности и ее замену, если хорошо представляете себе то, что задумали. Так, например, разрушительной будет операция замены группы байтов на другую группу, которая длиннее первой.

16. MOD (число, число)

FN V(число, число)

Эта функция дает остаток от деления первого числа на второе.

MOD (10, 3) = 1
MOD (66,16) = 2
MOD (125,35.5) = 18.5

Нижеприведенный пример показывает, как можно избежать попыток выполнения команды PLOT вне пределов экрана.

```
10 FOR n=0 TO 400
20   PLOT MOD(n, 256),MOD(n,176)
30 NEXT n
```

17. NUMBER (симв. строка)

FN N(симв. строка)

См. также CHAR$ (число)

Функция преобразует двухсимвольную строку в целое число от 0 до 65535. Эквивалентом в стандартном БЕЙСИКе является выражение:

```
LET num=256*CODE c$(1)+CODE c$(2)
```

Если символьная строка имеет более двух символов, выдается сообщение об ошибке "Invalid argument".

С помощью функции CHAR$ функция NUMBER может применяться для создания целочисленных массивов, в которых вместо чисел использованы их символьные эквиваленты.

18. OR (число, число)

FN O(число, число)

Эта функция произносится так же, как и обычное ключевое слово "OR", но имеет другое действие. В программе или при вводе отличается иным синтаксисом.

Функция дает результат побитной логической операции "ИЛИ" для двух чисел, каждое из которых находится в пределах от 0 до 65535. Если какой-либо бит включен в первом или втором числе, то в результате этот бит тоже включен (равен 1). Он будет равен 0 только если он выключен в обоих числах одновременно.

19. RNDM (число)

FN R(число)

Если "число" равно 0, то функция RNDM выдает случайное число от 0 до 1 - точно так же, как и функция RND стандартного БЕЙСИКа. Если же "число" не ноль, то функция выдает случайное целое число, лежащее в диапазоне от нуля до заданного "числа".

Эта функция работает в два с половиной раза быстрее, чем работало бы выражение RND*"число".

```
10 PLOT RNDM(255),RNDM(175)
20 GO TO 10
```

Оператор RANDOMIZE "число" устанавливает генерируемую псевдослучайную последовательность в определенное положение точно так же, как он делает это для функции RND в стандартном БЕЙСИКе.

20. SCRN$ (строка, столбец)

FN K$(строка, столбец)

Работает примерно так же, как и стандартная функция SCREEN$, за исключением того, что может распознавать и символы графики пользователя UDG. Кроме того, исправлена ошибка системного ПЗУ, которая отражается на работе функции SCREEN$. Об этой ошибке мы писали в недавно выпущенной "ИНФОРКОМом" книге "Элементарная графика". См. также статью "Ошибки ПЗУ" в этом номере "ZX-РЕВЮ".

Перед тем, как набрать нижеприведенный пример, дайте команду KEYWORDS 0. Программа распределит по экрану символы графики пользователя в виде случайного рисунка, а затем считает некоторые из них с экрана.

```
10 FOR a=USR "a" TO USR "u" + 7
20 POKE a, RND*255
30 NEXT a
40 PRINT "символы графики пользователя"
50 LET a$- =""
60 FOR c=1 to 31
70 LET a$ = a$ + SCRN$(0,c)
80 NEXT c
90 PRINT a$
```

Блочную графику "Спектрума" эта функция не распознает. Если Вам и это необходимо, то запрограммируйте некоторые символы графики пользователя так, чтобы они выглядели, как символы блочной графики.

Символы могут распознаваться только в том случае, если они изображены в

стандартном размере 8x8 пикселов (см. CSIZE).

21. SHIFT$ (число, строка)

FN Z$(число, строка)

SHIFT$ - многоцелевая функция для преобразования строковых переменных. Она имеет много разных режимов работы. Режим задается параметром "число" при вызове функции. Вот краткий обзор ее режимов.

1. Все символы строки преобразуются в верхний регистр (в прописные буквы).
2. Все символы преобразуются в нижний регистр (в строчные буквы).
3. Регистр всех символов меняется на противоположный.
4. Подавление управляющих кодов. Все символы, являющиеся управляющими кодами, за исключением символа CHR$ 13 (код ENTER) заменяются символом "точка" (".").
5. Подавление токенов ключевых слов. Символы CHR$ 128...255 заменяются символами 0...127. При этом управляющие коды, за исключением ENTER (CHR$ 13) заменяются символом ".".
6. Подавление токенов ключевых слов. Символы CHR$ 128...255 заменяются символами 0...127, при этой все управляющие коды заменяются символом ".".
7. Все ключевые слова преобразуются из токенизированной (однобайтной) формы в многобайтную (по байту на каждый символ).
8. Все ключевые слова преобразуются из формы с полным написанием в однобайтные токены. Регистр символов роли не играет. После каждого ключевого слова должен стоять небуквеный символ.
9. То же, что и предыдущий режим, но после ключевого слова может стоять любой символ.
10. То же, что и предыдущий режим, но не все ключевые слова должны быть набраны прописными литерами.
11. То же, что и режим 8, но все ключевые слова должны быть набраны прописными буквами.

SHIFT$1...SHIFT$3

Преобразования регистров.

Рассмотрим примеры:

SHIFT$ (1,"Basic") = "BASIC"

SHIFT$ (2,"Basic") = "basic"

SHIFT$ (3,"Basic") = "bASIC"

Обычное применение этих режимов - преобразование символьных строк, вводимых пользователем, перед сравнением с контрольной строкой в диалоговых программах.

```
100 INPUT i$: IF SHIFT$(1,1$) ="Y" THEN GO TO 200
```

Это поможет вам уйти от целой последовательности сравнений, таких, как

```
IF i$ = "Y" OR i$ = "y"
```

При работе с базами данных, эта функция может использоваться для того, чтобы предварительно конвертировать массив записей пользователя в верхний регистр, прежде чем давать команду SORT.

SHIFT$4...SHIFT$6

Подавление управляющих кодов и токенов ключевых слов.

Эти режимы, по-видимому, найдут широкое применение у тех пользователей, которые программируют в машинных кодах. Так, при просмотре содержимого памяти компьютера, Вам может быть захочется распечатать содержимое ячеек командой

```
PRINT CHR$(PEEK address)
```

Очень скоро по этой команде Вы получите сообщение об ошибке "Invalid colour". Это произойдет как только вы попытаетесь распечатать непечатный символ.

Например, последовательность 17, 200 будет интерпретироваться, как CHR$17; CHR$200, а это в переводе с машинного языка на БЕЙСИК означает PAPER 200. Компьютер отреагирует сообщением об ошибке.

Сам формат печати при этом будет выглядеть весьма неопрятно, поскольку символы выше 127 могут быть распечатаны, как символы UDG, как символы блочной графики и как токены ключевых слов, имеющие самую разную длину.

Функция SHIFT$ позволяет справиться с этой проблемой, подавляя неприятные эффекты. В течение нескольких минут Вы сможете "прощупать" память компьютера в поисках таблиц данных, сообщений и списков ключевых слов.

```
100 FOR n=1 TO 65535 STEP 704
110 PRINT SHIFT$(6,MEMORY$())(n TO n+703)
120 PAUSE 0: CLS
130 NEXT n
```

Если область памяти, которую вы сканируете, содержит не машинный код, а БЕЙСИК-программу, то может быть Вам нецелесообразно отключать изображение токенов ключевых слов и достаточно только подавить управляющие коды режимом SHIFT$4.

SHIFT$7

Преобразование токенов в полную символьную запись.

Начнем с примеров:

```
10 LET a$ = " THEN NOT":
    REM это токены
20 PRINT a$, LEN a$:
    REM LEN=2
30 LET t$ = SHIFT$(7,a$)
40 PRINT t$, LEN t$:
    REM LEN=9
```

Эта функция должна быть полезной для тех, кто работает с принтером. Если принтер подключен не через стандартный "Синклеровский" интерфейс, то он не сможет воспроизводить на печать токены ключевых слов БЕЙСИКа, поскольку он о них ничего не знает.

Подобная конверсия поможет вам получать распечатки ваших БЕЙСИК-программ.

SHIFT$8...SHIFT$11

Преобразование ключевых слов из полной формы записи в токенизированную форму.

Эта функция преобразует все символьные последовательности, которые являются ключевыми словами БЕЙСИКа в токенизированную форму. Ключевое слово не будет распознано, если непосредственно перед ним стоит какая-либо буква. Что же касается символа, стоящего непосредственно за ключевым словом, то его влияние зависит от того, какой конкретно режим был избран (см. выше).

Эта функция может пригодиться в том случае, если вы примете БЕЙСИК-программу через внешний порт или через сеть от компьютера другой системы. Конвертировав записанные символами ASCII ключевые слова в токены и подправив синтаксис программы под свой "Спектрум", вы сэкономите массу времени, т. к. вам не придется набирать текст программы вручную.

22. SINE (число)

FN S(число)

Это модифицированная функция "синус". Она дает менее точный результат по сравнению с функцией SIN стандартного БЕЙСИКа, но зато работает в шесть раз быстрее.

Если для математических приложения она, может быть и не годится, зато для работы с векторной графикой, при расчете координат проекций на экране она будет хороша.

23. STRING$ (число, строка)

FN S$ (число, строка)

Эта функция дает строковую переменную, состоящую из параметра "строка", повторенного столько раз, каково значение параметра "число".

```
STRING$ (32,"-") = "--.....--"
```

(32 знака)

```
STRING$ (4,"AB") = "ABABABAB"
PRINT STRING$(704,"X")
```

- печатает экран, заполненный символами "X".

```
PRINT STRING$(3,"A"+CHR$ 13)
```

печатает: A
A
A

Если вам нужно сгенерировать строку, состоящую из более, чем 14 повторяющихся символов, то использовать STRING$ удобнее, чем вводить символы от руки. Кроме того, функция STRING$ работает быстрее, чем цикл FOR...NEXT, который тоже может быть применен для создания длинной регулярной строки.

Эта функция может применяться в Бета-Бейсике для заполнения блоков оперативной памяти информацией, например для установки необходимых экранных атрибутов. Для этой цели она используется совместно с функцией POKE.

24. TIME$ ()

FN T$()

См. также команду CLOCK.

Эта функция выдает текущие показания встроенных часов (если они были инициализированы в Бета-Бейсике). Если Вы несколько раз повторите команду PRINT TIME$(), то всякий раз получите разный результат. В программах невредно передать показания часов какой-либо переменной и, тем самым, "заморозить" полученный отсчет.

```
100 CLOCK 1
110 LET n$ = TIME$(): PRINT n$
120 PRINT "HOUrS= "; n$(1 TO 2);"Mins = ";n$(4 TO 5)
130 GO TO 110
```

Так можно встроить в программу контроль за временем исполнения программы пользователя. Подробности смотрите в разделе, посвященном описанию команды CLOCK.

25. USING$ (строка, число)

FN U$(строка, число)

См. также команду USING.

Функция конвертирует "число" в строковую переменную в формате, заданном параметром "строка". Вы можете задать количество изображаемых символов до десятичной точки и после. Ключевое слово USING, расположенное на клавише U обеспечивает то же самое в команде PRINT. В отличие от него, функция USING$ позволяет не только распечатать полученный результат, но и запомнить его. Она может быть использована не только с командой PRINT, но и с другими командами, допускающими работу со строковыми переменными. Более подробное описание смотрите в разделе, посвященном команде USING.

26. XOR (число, число)

FN X(число, число)

Функция выдает результат побитной операции "ИСКЛЮЧАЮЩЕЕ ИЛИ" для двух чисел, которые должны быть в пределах от 0 до 65535. Если какой-то бит и в первом числе и во втором равны между собой, то в результате этот бит будет равен нулю. Если же они противоположны, то в результате он будет равен единице.

ПРИЛОЖЕНИЕ 1

Ключевые слова Бета-Бейсика. Версия 3.0.

КОД	КЛАВИША	ТОКЕН
128	8	KEYWORDS
129	1	DEF PROC
130	2	PROC

131	3	END PROC
132	4	RENUM
133	5	WINDOW
134	6	AUTO
135	7	DELETE
136	Shift 7	REF
137	Shift 6	JOIN
138	Shift 5	EDIT
139	Shift 4	KEYIN
140	Shift 3	LOCAL
141	Shift 2	DEFAULT
142	Shift 1	DEF KEY
143	Shift 8	CSIZE
144	A	ALTER
145	B	-------
146	C	CLOCK
147	D	DO
148	E	ELSE
149	F	FILL
150	G	GET
151	H	-------
152	I	EXIT IF
153	J	WHILE
154	K	UHTIL
155	L	LOOP
156	M	SORT
157	N	ON ERROR
158	O	ON
159	P	DPOKE
160	Q	POP
161	R	ROLL
162	S	SCROLL
163	T	TRACE
164	U	USING

Примечание:

Для того, чтобы вернуться к стандартному для "Спектрума" значению вышеуказанных кодов, нужно перейти в режим работы KEYWORDS 0.

ПРИЛОЖЕНИЕ 2.

Сообщения об ошибках Бета-Бейсика, версия 3.0.

G No room for line

Заданные параметры при перенумерации строк приводят к тому, что в результате перенумерации появляется строки с номерами из диапазона, не подлежащего перенумерации или появляются строки с номерами больше 9999.

Ошибка проявляется при работе команды RENUM.

S Missing LOOP

Оператор цикла по условию DO WHILE, DO UNTIL или оператор выхода из цикла EXIT IF не могут найти оператора конца цикла LOOP.

Ошибка проявляется при работе операторов DO WHILE, DO UNTIL, EXIT IF.

T LOOP without DO

В программе присутствует оператор LOOP, но нет соответствующего ему оператора DO.

U No such line

В программе был использован оператор DELETE с указанием в качестве параметра номера строки, которой в программе нет.

Ошибка проявляется при работе оператора DELETE.

V No POP data

При попытке выполнить оператор POP оказывается, что стек GO SUB/ DO-LOOP/PROC - пуст. Это означает, что в данный момент времени не исполняются ни подпрограммы GO SUB, ни циклы DO-LOOP, ни процедуры PROC.

Ошибка проявляется при исполнении оператора POP.

W Missing DEF PROC

В программе была попытка исполнить процедуру, которая ранее не была определена оператором DEF PROC. То же происходит, если различаются имена процедуры при задании и при вызове. Ошибка может возникать, если встретился оператор END PROC, а процедура ранее не была объявлена через DEF PROC.

Ошибка проявляется при вызове процедур и при исполнении операторов END PROC и LOCAL.

X No END PROC

Программа во время работы пытается обойти объявление процедуры, но не может найти оператора END PROC, который соответствовал бы DEF PROC.

Ошибка проявляется при встрече оператора DEF PROC.

ПРИЛОЖЕНИЕ 3

Коды ошибок.

Ниже приведен список кодов, которые записываются в переменную ERROR по команде ON ERROR. Список состоит из трех разделов. В первом разделе перечислены состояния, связанные со стандартным БЕЙСИКОМ "Спектрума". Во втором разделе - ошибки БЕТА-БЕЙСИКА. В третьем разделе - ошибки, связанные с ИНТЕРФЕЙСОМ-1.

Примечание: Коды 0 и 9 (а они фактически ошибками не являются) не перехватываются оператором ON ERROR.

1. Для стандартного БЕЙСИКа.

Значение ERROR	Код ошибки	Сообщение
0	0	O.K.
1	1	NEXT without FOR
2	2	variable not found
3	3	Subscript wrong
4	4	Out of memory
5	5	Out of screen
6	6	Number too big
7	7	RETURN without GOSUB
8	8	End of file

9	9	Stop statement
10	A	Invalid argument
11	B	Integer out of range
12	C	Nonsense in Basic
13	D	BREAK-CONT repeats
14	E	Out of DATA
15	F	Invalid file name
16	G	No room for line
17	H	STOP in INPUT
18	I	FOR without NEXT
19	J	Invalid I/O device
20	K	Invalid colour
21	L	BREAK into program
22	M	RAMTOP no good
23	N	Statement lost
24	O	Invalid stream
25	P	FN without DEF
26	Q	Parameter error
27	R	Tape loading error

2. Для БЕТА-БЕЙСИКа

28	S	Missing LOOP
29	T	LOOP without DO
30	U	No such line
31	V	No POP data
32	W	Missing DEF PROC
33	X	No END PROC

3. Для ИНТЕРФЕЙСа-1.

43	b	Program finished
44	c	Nonsense in BASIC
45	d	Invalid stream number
46	e	Invalid device expression
47	f	Invalid name
48	e	Invalid drive number
49	h	Invalid station number
50	i	Missing name
51	j	Missing station number
52	k	Missing drive number
53	l	Missing baud rate
54	m	Header mismatch error
55	n	Stream already open
56	o	Writing to a "read" file
57	p	Reading a "write" file
58	q	Drive "write" protected
59	r	Microdrive full
60	s	Microdrive not present
61	t	File not found
62	u	Hook code error
63	v	CODE error
64	w	MERGE error
65	x	Verification has failed
66	y	Wrong file type

На этом мы заканчиваем печать инструктивных материалов, посвященных диалектам БЕЙСИКА, выпущенным фирмой BETASOFT. За два года мы рассмотрели BETA-BASIC (версии 1.0, 1.8 и 3.0.).

Существует и еще более мощная версия 4.0, которая поддерживает работу с дисковой операционной системой. К сожалению, технической документации по этой версии у нас пока нет, а потому мы будем очень признательны тем из наших читателей, которые смогут такую документацию предоставить (на языке оригинала). Условия приобретения обсуждаются, в этом случае мы сможем довести эту информацию до самых широких кругов.

ЗАЩИТА ПРОГРАММ

Сегодня мы заканчиваем печать третьего тома книги В.С. Михайленко, посвященной вопросам защиты программ для "Синклер"-совместимых компьютеров и переходим к четвертому тому, который написан в соавторстве с экспертом из Белорусского Государственного Университета (БГУ) А. К. Туровичем.

Полностью печать статей данного цикла будет завершена в 1993 году.

Продолжение.
(Начало: 9-16, 53-60, 97-104, 141-146, 185-192).

1.2 Смещение системной переменной PROG.

Многие из Вас, вероятно, не раз убеждались в справедливости принципа, "чем проще - тем надежнее". Этот принцип был известен издавна и не раз подтверждал себя на практике. Актуальным он является и для нас, потому что темой нашего разговора будет метод, основанный на смещении системной переменной PROG. Метод, на первый взгляд достаточно простой, но, тем не менее, достаточно эффективный. Рассмотрим более подробно принцип его применения для защиты компьютерных программ.

Как Вам уже, вероятно, известно, Бейсик, в стандартном "Спектруме (без подключенной периферии) начинается с адреса 23755. Об этой свидетельствует содержимое системной переменной PROG (23635). Таким образом, практически всегда Бейсик в компьютере начинается с одного и того же адреса памяти. Однако, такое положение вещей достаточно легко изменить, если осуществить изменение системной переменной PROG. Рассмотрим, что это нам дает.

Предположим, вы разработали новый загрузчик в кодах и хотели бы затруднить его прочтение и просмотр без Вашего ведома. Для этого достаточно хорошо подходит данный метод защиты.

Для начала вы создаете специальную программу в машинных кодах, которая осуществляет изменение системной переменной PROG. После этого Вам понадобится совместить ее с программой на Бейсике (о том, как это сделать, было подробно написано в т.1 гл.1). Конечно, можно достаточно просто изменить содержимое PROG и из Бейсика путем применения POKE, однако предпочтительней закамуфлировать выполненные операции, а еще лучше замаскировать адрес старта программы в кодах одним из методов, предложенных в т.1 гл. 4 (Напомню что основное предпочтение отдавалось там изменению содержимого числового значения, расположенного после управляющего кода 14). Заканчивает данную программу на Бейсике команда LOAD"".

Как Вы уже, вероятно, догадались, данная Бейсик-программа служит исключительно для подготовки к загрузке Вашей специальной программы-загрузчика. Причем Ваш загрузчик в машинных кодах ассемблирован таким образом, чтобы работать только в новой области (после изменения системной переменной PROG). Адрес старта загрузчика, вполне естественно, тоже будет замаскирован и будет указывать на точку расположения вашей исходной процедуры после изменения системной переменной. Теперь Вам осталось только совместить исходную программу в машинных кодах с новой Бейсик-программой и после выяснения адреса реального старта осуществить необходимые изменения чисел после функции RANDOMIZE USR.

Давайте теперь рассмотрим, как работает данная защита. После проведения вышеописанных работ мы имеем две Бейсик-программы, в каждую из которых встроен блок машинных кодов. Причем процедура в кодах, встроенная в первую Бейсик-программу, осуществляет изменение системной переменной PROG, в то время как вторая процедура в кодах, встроенная во вторую Бейсик-программу, является ничем иным, как исходной программой-загрузчиком, которую Вы и собираетесь защитить от несанкционированного

просмотра.

После загрузки первой Бейсик-программы, она автоматически запускается и изменяет системную переменную PROG так, чтобы следующая Бейсик-программа загружалась уже в новую область. Такое изменение дает нам исключительную возможность правильно запустить Вашу программу-загрузчик в машинных кодах. Это объясняется тем, что ваша программа-загрузчик ассемблирована под новое значение системной переменной PROG и не будет работать, если не выполнить вышеописанных изменений.

Карта памяти компьютера до и после изменения системной переменной PROG показана на рис. 1.

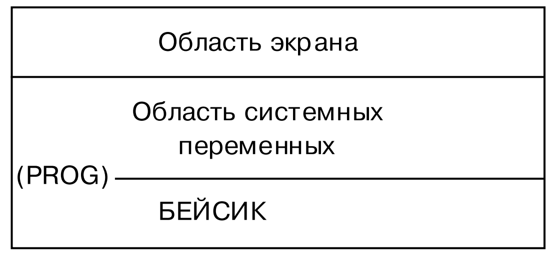


Рис. 1 а

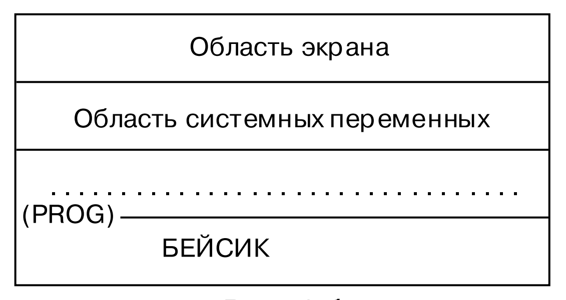


Рис. 1 б

Случай а) показывает вариант, когда Бейсик находится сразу после области системных переменных, т.е. начиная с адреса 23755. Случай б) рассматривает вариант изменения значения системной переменной PROG, т.е. между областью системных переменных и Бейсиком существует незаполненное полезной информацией пространство.

Примечание: Разумеется, изменение системной переменной PROG само по себе уменьшает объем памяти, доступной для пользователя, однако это бывает оправдано, например, когда мы строим на этом методе защиту.

Любопытно, что системная переменная PROG изменяется принудительно, в случае подключения INTERFACE 1. в этом случае между областью системных переменных и Бейсиком размещаются системные переменные, необходиные для работы 8K ПЗУ INTERFACE 1.

На рисунке взятая в скобки надпись системной переменной PROG означает, что Бейсик начинается с ячейки памяти, адрес которой находится в системной переменной PROG.

Теперь вы видите, что если попробовать загрузить второй Бейсик-файл не изменяя PROG, то программа в кодах, размещенная там, не будет работать, аналогично, как не будет работать и обыкновенная программа в кодах, перемещенная из места, для которого она ассемблирована, в какое-либо иное место памяти. Для программистов, работающих в машинных кодах причина неработоспособности в подобном случае очевидна - не совладают адреса переходов при обычной адресации.

Небольшая историческая справка.

Когда взлом компьютерных программ только начинался, во многих программах Билла Гильберта, да и других "хаккеров" можно было встретить строку приблизительно следующего содержания:

```
0 LIST USR (PEEK 23635 + 256*PEEK 23636 +17)
```

которая собственно и должна была запускать встроенную программу в машинных

кодах. Очевидно, что LIST USR - это лишь одна из разновидностей команды RANDOMIZE USR (об этом уже было записано в главе "новейшие достижения защиты"). Как видно, программа в кодах запускалась с адреса на 17 байтов большего, чем тот, на который указывала системная переменная PROG. Четыре байта уходили на номер и длину строки, еще один - на код оператора REM. А остальное место до начала программы Гильберт любил заполнять своими инициалами. Удобство такой записи адреса старта процедуры в кодах заключалось в том, что она сама находила, где находится Бейсик-программа. Это было необходимо ввиду широкого распространения в то время ИНТЕРФЕЙСА 1 и микродрайвов, которые изменяли значение PROG. Кроме того, такая запись затрудняла пользователю прочтение реального адреса старта машинокодовой процедуры.

Однако, позднее на основе этого появилось достаточно любопытное направление защиты программ, описание которого было приведено выше. Я надеюсь, что мои рекомендации не только помогут читателям лучше прояснить работу своего компьютера, но и пригодятся в повседневной работе при разработке своих программ.

1.3 Кодирование и декодирование блоков машинных кодов.

Одной из разновидностей защиты программ в кодах является кодирование этих процедур. Под кодированием понимается изменение истинных значений программы в кодах с целью дестабилизации ее работы в нераскодированнои виде. Для обеспечения нормального функционирования программы в кодах ее необходимо подвергнуть раскодированию.

Одним из самых простых методов кодирования является изменение содержимого программы с использованием команды LDIR. Рассмотрим, как работает процедура, обеспечивающая правильное выполнение данной защиты.

Не вдаваясь в подробности, хочу напомнить читателям, что инструкция LDIR осуществляет перенос блоков кодов из одного места памяти в другое. Все необходимые значения для выполнения данной операции задаются заранее и заносятся в регистры микропроцессора. Для тех, кто интересуется работой данной команды процессора Z80 более подробно рекомендую книгу "ИНОФОРКОМа" "Первые шаги в машинных кодах".

Итак, каким же образом действует данная защита. Предположим, что исходная программа содержит два блока в машинных кодах. Один из них достаточно большой (порядка нескольких десятков килобайт), а второй небольших размеров (порядка нескольких килобайт или даже нескольких сот байтов).

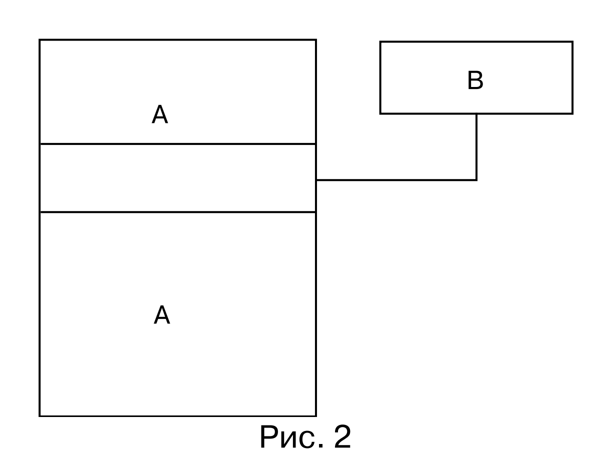


Рис. 2

На рисунке изображены блоки кодов, из которых состоит исходная программа.

Стрелкой показано, что после загрузки второй блок занимает заранее отведенное ему место в памяти, после чего программа может нормально функционировать. Перемещение блоков кодов из одного места памяти в другое осуществляется с использованиеи инструкции LDIR.

На рис. 2 изображена технология осуществления защиты данного типа защиты. Сразу после запуска программы осуществляется перенос блоков кодов из одного места памяти в другое, после которого программа может нормально функционировать.

Разумеется, в каждом конкретном случае эта защита может иметь определенные разновидности. В частности, исходный программный файл может состоять не из двух, а из гораздо большего числа блоков и перенос может осуществляться не для целого блока, а лишь какой-то его части. Если Вам ясен сам принцип защиты, то Вам не составит труда разобраться с его каждым конкретным применением.

Одним из примеров программы, использующей данный принцип, защиты может служить программа GREEN BERET, загрузчик которой был достаточно подробно описан в четвертой главе второго тома.

Следующий тип кодирования, который я хочу предложить Вашему вниманию, тоже получил распространение. Это объясняется тем, что в данном случае нам не требуется догружать какие-либо дополнительные блоки, чтобы восстановить правильное содержимое исходной программы - достаточно использовать специальную программу декодирования, которая используя специальный алгоритм воссоздаст из загруженного блока кодов содержимое исходной программы.

Рассмотрим более подробно, как этого можно добиться.

Одним из наиболее широко используемых приемов является самая обыкновенная инверсия содержимого ячеек памяти.

Как Вы уже вероятно знаете, любое значение, содержащееся в определенной ячейке памяти можно представить десятиричным значением от 0 до 255, шестнадцатиричным значением от 0 до FF или двоичным значением 00000000 до 11111111. Для программиста все эти системы счисления альтернативны, хотя известно, что компьютер оперирует двоичными значениями числа, в то время как при программировании принято использовать шестнадцатиричную систему счисления (шестнадцатиричная система счисления - HEX система - является профессиональной системой счисления для программистов). Так вот, любое шестнадцатиричное число можно представить в двоичном виде, используя специальные таблицы. А процесс инверсии очень легко понять, используя двоичное представление. Инвертировать байт - это значит изменить содержимое каждого его бита на противоположное.

Например, байт 00 после инверсии превращается в байт FF:

0000 0000 - байт 00H представленный в двоичном виде;

1111 1111 - после инверсии все значения данного байта заменяются на противоположные. Шестнадцатиричное значение данного двоичного числа FF.

Аналогично, байт 01010101 после инверсии преобразуется в 10101010 и т.д.

Как видите, кодирование инвертированием может применяться достаточно успешно, поскольку байты загружаемого блока не смогут правильно обрабатываться микропроцессором без соответствующего преобразования. Для того, чтобы программа заработала правильно, необходимо перед ее запуском снова инвертировать эти байты, чтобы все стало на свои места.

Однако кодирование инверсией, как Вы могли убедиться, является достаточно примитивным и поэтому настало время рассмотреть систему защиты, применяемую в большинстве фирменных программ (ART STUDIO, ATIC ATAC, NIGHT SHADE, THE WORD).

Рассмотрим эту систему зашифровки на примере программы "ART STUDIO".

В данном случае кодирование представляет собой систему достаточно простого типа, делающего невозможным правильную работу программы. Для расшифровки применяется специальная декодирующая процедура, которая находится в той же программе и, что очень важно, не закодирована. В данном случае (впрочем, как и во всех рассмотренных ранее) кодирование просто затрудняет доступ к тексту программы, после того, как все предшествующе защиты устранены и программа считана без автозапуска.

Несмотря на то, что бейсиковская часть программы практически не защищена, необходимо достаточно внимательно изучить систему декодирования прежде, чем осуществлять эксперименты с запуском процедуры на выполнение.

Поскольку запуск программы начинается с адреса 26000, исследуем команды, расположенные начиная с этого адреса:

```
26000 JP 26024
```

Как видим, сразу осуществляется переход к процедуре декодирования. (В дальнейшем мы рассмотрим с Вами и процедуру кодирования, размещенную в данной программе в адресах 26003...26021).

```
26034 LD HL,26049
26027 PUSH HL
26028 LD HL,25049
26031 LD DE,26719
26034 LD A (HL)
26035 SUB 34
26037 RLCA
26038 XOR #CC
26040 LD (HL),A
26041 INC HL
26042 OR A
26043 SBC HL,DE
26045 ADD HL,DE
26046 JP NZ,26034
26048 RET
```

Данная процедура сначала помещает в стек значение адреса 26049 - сюда программа перейдет после выполнения команды RET.

После этого и начинается процесс декодирования; в регистр HL снова загружается адрес 26049 - как начало декодируемого блока. Затем в цикле декодируются последовательно байты загружаемого блока, причем инструкции:

```
SUB 34
RLCA
XOR #CC
```

являются ключем, с помощью которого расшифровывается эта часть программы. В ходе работы проверяется условие достижения адреса 27719, как последнего декодируемого байта (это значение содержится в DE).

Выполнение инструкции RET приводит не к возврату в Бейсик, а к переходу на адрес, который был последним занесен в стек - в нашем случае 26049. Следовательно, по этой инструкции происходит запуск на исполнение расшифрованного блока.

Следует отметить, что инструкции, осуществляющие дешифрацию, в ходе своей работы не теряют ни одного бита. Это необходимое условие для данного типа программ, поскольку кодированию и декодированию может подвергнуться практически любой байт от 00 до FF.

Теперь рассмотрим, каким образом можно осуществить декодирование данной программа, чтобы после возврата в Бейсик подробно изучить новообразующуюся часть программы. Для этого нам понадобится перестроить работу исходной программы так, чтобы на стек не попало значение 26049, а там оставалось последним значение возврата в бейсик-программу.

Для выполнения поставленной цели существуют два пути:

- либо уничтожить команду PUSH HL, заслав вместо нее код NOP, т.е. ноль.

- либо запустить программу после помещения на стек вышеуказанных значений, командой RANDOMIZE USR 26028. В этом случае мы возвратимся в Бейсик по выполнении инструкции RET.

Теоретически оба метода взаимозаменяемы и могут быть применены равноправно. Однако, на практике оказывается, что это не так. В частности, в программе ART STUDIO существует блок программы, проверяющий сохранность значений ячеек памяти в месте расположения декодирующей процедуры. Если он обнаружит изменение содержимого ячеек памяти, то программа зависнет.

Процедура проверки находится начиная с адреса 26283:

```
26263 LD H,A
26284 LD L,A
26285 PUSH HL
26292 LD A,(26024)
26295 CP #21
26297 RET NZ
```

```
26298 LD HL,(26025)
26301 OR A
26302 LD DE,26049
26305 SBC HL,DE
86307 RET NZ
26308 LD A,(26027)
26311 CP #E5
26313 RET NZ
26314 POP HL
26315 RET
```

Рассмотрим более подробно, как работает данная процедура.

Для начала мы помещаем 0 на стек, используя то, что содержимое аккумулятора равно 0. После этого осуществляется проверка байтов 26024-26027 (включительно) и, если обнаруживается несовпадение с тем, что там ожидал найти автор программы, то происходит возврат на 00, а это ничто иное, как перезапуск компьютера. Если же проверка прошла успешно, мы снимаем ноль со стека и возвращаемся в вызвавшую данную подпрограмму процедуру.

Теперь Вы видите, что менять содержимое ячейки POKE 26027,0 достаточно рискованно. Поэтому в данном случае лучше воспользоваться вторым предложенным вариантом, а на будущее запомнить, что возможность проверки работы защиты - достаточно частое явление в компьютерных программах.

Рассмотренные нами три примера, разумеется, не исчерпывают всех возможных вариантов кодирования. Каждый программист старается разработать свой собственный метод, как можно более изощренный. Однако, при подобной разработке Вам необходимо учитывать, что инструкции, осуществляющие шифрацию и дешифрацию, не должны терять в ходе своей работы ни одного бита. В программе ART STUDIO вычитание производится по модулю 256 и для двух разных входных данных результаты тоже различны. RLCA заменяет значение битов 7,6,3,2 на противоположные.

Другими операторами, имеющими аналогичные свойства, являются ADD, INC, DEC, RRCA, CPL и др.

Однако, в данном случае нельзя использовать функции OR или AND, поскольку Вам не удастся правильно восстановить содержимое исходной программы.

1.4 Новые POKES.

Многие программисты, длительное время работающие с компьютером, наверняка изучили большинство POKES, обычно применяемых для защиты компьютерных программ. Многие считают малоперспективным разрабатывать новые приемы защиты, основанные на этом направлении и пытаются создавать более изощренные приемы. Тем не менее, несмотря на свою давнюю историю, этот метод защиты компьютерных программ не следует предавать забвению.

Несколько новых адресов, благодаря использованию которых можно защитить информацию от несанкционированного просмотра, помогли данному методу подняться на новые рубежи и составить конкуренцию традиционно используемым в современных игровых программах приемам защиты.

Метод засылки в определенные ячейки памяти измененных значений (в большинстве случаев этими ячейками являются системные переменные) начал применяться с самого начала появления защиты компьютерных программ для "Спектрума". Он применялся как для защиты от нажатия клавиши "BREAK", так и от произвольного листинга. (См. т.1). Следует отметить, что в дальнейшем использование данной методики было сильно ограничено ввиду незначительного количества ячеек памяти, которые приводили к защитному эффекту. Это привело к тому, что практически вся информация о POKES быстро стала широким достоянием большого числа "хаккеров". По этой причине информация о новых POKES, имеющих "защитные" функции является тем козырем, который даст вам возможность охладить неумеренный пыл юных взломщиков. В этой статье я приведу информацию о двух ячейках памяти, изменяя содержимое которых, можно получить либо оригинальную защиту от листинга, либо защитить работающую программу от непредусмотренной остановки

нажатием клавиши "BREAK". Несмотря на то, что эти методы различны по принципу действия, их объединяет то, что изменение содержимого ячеек памяти происходит в области системных переменных. Рассмотрим каждый из них более подробно.

1. POKE 23743,80 приводит к тому, что мы не можем получить на экране никакой информации, в том числе и листинга. Это объясняется тем, что мы изменили адрес программы вывода на основной экран компьютера. Тем не менее, адрес программы вывода с клавиатуры в нижнюю часть экрана остался прежним и поэтому мы можем вызвать в командную строку любую из строк исходной программы командой EDIT. Такая возможность несомненно является достаточно значительным дефектом эти защиты, однако это дело поправимое, поскольку для защиты от подобных ухищрений Вы можете аналогичным образом изменить и адрес программы вывода для этого канала.

В заключение лишь следует добавить, что для нормализации работы компьютера Вам необходимо набрать:

```
POKE 23743,83
```

Эти изменения нормализуют работу канала и теперь Вы можете без труда ознакомиться с листингом.

Данный метод защиты был обнаружен мной в программе MASBLAST -1990.

2. POKE 23613, PEEK (23700)-5 создает защиту от нажатия клавиши "BREAK" и приводит вначале к зависанию компьютера, а через несколько секунд к самосбросу.

Данный метод основан на изменении содержимого одной из системных переменных, ответственных за адрес в аппаратном стеке, используемый как адрес возврата при ошибке. Изменив предлагаемым образом содержимое ячейки, Вы при нажатии клавиши "BREAK" попадаете на мнимую подпрограмму обработки ошибки, которая и приводит ко всем вышеописанным результатам.

Этот метод защиты, к сожалению, нельзя разблокировать во время его работы и для успешного его преодоления вам придется загружать программу без автозапуска одним из предложенных ранее методов.

Как видите, данные методы защиты основаны на использовании хорошо известных приемов POKE и я уверен, что возможности данной методики еще далеко не исчерпаны и в будущем свое слово здесь скажете Вы, дорогие читатели.

1.5 Метод нулевых строк - новые возможности.

В данной статье разобран оригинальный метод защиты компьютерных программ к "Спектрум"-совместимым компьютерам. Подробные комментарии позволяют использовать этот материал не только как полезную процедуру, но и как пособие тем, кто самостоятельно изучает программирование на языке АССЕМБЛЕРА.

В первом томе (см. стр. 13) Вашему вниманию была предложена статья "Универсальная система защиты - метод нулевых строк". Несмотря на определенные достоинства, эта программа обладает целым рядом недостатков, которые в некоторых случаях могут нарушить работу Вашей исходной программы на Бейсике. Постараюсь объяснить причину возникающих в ходе работе программы "зануления" неточностей.

Как уже вероятно убедились пользователи, работавшие с моей программой, она практически всегда справляется со своей задачей. Однако, необходимо заметить, что программа не учитывает некотрых особенности Бейсик-строки "Спектрума" и поэтому существует вероятность ошибочного "зануления". Несмотря на то, что в большинстве случаев программа работает нормально, теоретически нельзя не учитывать возможность ошибки. Попробуем разобраться, почему это может произойти.

Вам Вам уже, вероятно, известно, все числа в Бейсике "Спектрума" представлены в пятибайтной интегральной форме. Это достаточно своеобразное представление и необходимо оно для правильной работы встроенного калькулятора. Существует несколько различных форм данного пятибайтного представления, в зависимости от того, является ли исходное число действительным либо целым (для целых чисел форма представления также различна - все зависит от того, является ли оно положительным или отрицательным). Для

тех, кто хочет более подробно разобраться в этом вопросе, рекомендую читать "Первые шаги в машинных кодах".

Я же, не вдаваясь в подробности, хочу отметить, что в этой пятибайтной последовательности чисел вполне может встретиться и число "13", однако в случае наличия его в пятибайтном числовом стринге, оно не должно анализироваться программой, как код "ENTER ", поскольку данный стринг является единым целым (компьютер определяет это по обнаружению управляющего кода "14"). Во всех остальных случаях число "13" должно восприниматься компьютером как код "ENTER".

Предложенная в первом томе программа не учитывала, что число "13" может содержаться в такой пятибайтной форме чисел и поэтому вполне могло произойти ошибочное "зануление". В самой деле, программа, обнаружив подобное число, обязательно превратило бы в ноль следующие за этим числом два байта, в то время как делать этого не следует.

Второй обнаруженный недочет состоит в том факте, что моя программа начинает анализ Бейсика с адреса 23755, в то время как это не всегда корректно, поскольку в некоторых случаях область Бейсика может сдвигаться "вверх" (например при подключении периферии). Точно определить адрес начала области Бейсика нам поможет содержимое системной переменной "PROG".

С момента опубликования вышеуказанной статьи я продолжал работу над данной темой и пришел к ряду выводов, с некоторыми из которых хочу поделиться с читателями. Разработанная мной программа не "зануляет" первую строку Вашей программы, что несомненно осложняет ее использование. Кроме этого, стоит отметить, что решение, когда программа "зануления" располагается после исходной программы на Бейсике, не совсем удачно, так как:

- по-первых, выполнение программы "зануления" осуществляется на Бейсике, что существенно сказывается на ее быстродействии;

- во-вторых, почти всегда придется "занулять" все строки программы, а это не всегда необходимо;

- в-третьих, бывают случаи, когда предпочтительней использовать процедуру в машинных кодах, поскольку после работы Бейсик-программы "зануления" ее необходимо уничтожить, в то время, как программа в машинных кодах в этом не нуждается.

Вышеописанные недостатки делают проблематичным использование моей исходной программы в некоторых случаях. Однако, мне удалось составить программу, которая не имеет вышеописанных недостатков. Рассмотрим ее более подробно.

Основным требованием к такого рода программе было то, чтобы она могла загружаться в любую область памяти без потери работоспособности. Это объясняется тем, что в некоторых случаях область памяти, для которой данная программа была ассемблирована, бывает занята и, чтобы обеспечить удобную эксплуатацию подобной процедуры, необходимо иметь возможность загружать ее в произвольную область ОЗУ с сохранением всех выполняемых функций. Подобный эффект становится возможным благодаря использованию относительной адресации. Этот вид адресации может использоваться командами как условного, так и безусловного переходов. В таком случае они состоят из двух байтов: первый байт содержит код операции, а второй - смещение в двоичной дополнительной системе. Действительный адрес получается прибавлением смещения к текущему показанию программного счетчика. Преимуществами относительной адресации перед абсолютной являются:

- команда занимает в памяти меньше места на один байт;

- программа становится перемещаемой, то есть не зависит от своего места расположения в памяти.

Полученная программа приведена в Листинге 1. Рассмотрим принцип ее действия.

Листинг 1.

```
Programing by Michailenko Vadim.
All rights reserved. Mensk 1991.
Michailenko Vadim driver system
```

```
for "EDITAS-48" files special for "INFORCOM".
10              ORG 64130
20              LD BC,9990
30              LD HL,(23635)
40              XOR A
50              LD HL,(A)
60              INC HL
70              LD (HL),A
80    ;     ----------------------------
90    NEXT      INC HL
100             LD E,(HL)
110             INC HL
120             LD D,(HL)
130             ADD HL,DE
140             LD A,(HL)
150             CP 13
160             RET NZ
170             INC HL
160             LD A,(HL)
190             CP B
200             JR Z,FINAL
210   ;     ----------------------------
220   CONT      XOR A
230             LD (HL),0
240             INC HL
250             LD (HL),0
260             JR NEXT
270   ;     ----------------------------
280   FINAL     INC HL
290             LD A,(HL)
300             CP C
310             RET Z
320             DEC HL
330             JR CONT
340             END
```

Сразу после начала работы в регистр BC заносится номер строки программы, до которой необходимо вести "зануление". Это значение можно изменять. Таким образом, Вы сможете осуществить "зануление" не всей исходной программы, а лишь определенной ее части. О том, как заменить число 9990 на другое, будет написан ниже.

После этого, по содержимому системной переменной PROG мы узнаем начало области Бейсика и осуществляем "зануление" первой программной строки.

Команда XOR A - простейший способ обнуления аккумулятора, ставший стандартным.

Это был подготовительный этап в работе программы - далее следует работа в циклическом режиме. По содержимому двух байтов, следующих за номером строки, программа определяет длину данной строки и, следовательно, сразу же может определить начало следующей. Теперь нобходимо проверить, не имеет ли данная строка программы номер 9990, который свидетельствовал бы об окончании работы, и, если номер совпадает, то процедура заканчивает свою работу и осуществляет возврат в Бейсик. В случае, если необходимый номер еще не достигнут, программа "зануляет" номер текущей строки и осуществляет возврат к началу цикла, после чего процесс повторяется.

Данная процедура имеет подстраховку для забывчивых пользователей. Если Вы обратили внимание, то возврат в Бейсик предусмотрен и после команды сравнения CP 13. Это может пригодиться, если Вы забыли создать строку с контрольным номером. В этом случае процедура "занулит" все строки программы и возвратится в Бейсик, когда обнаружит окончание исходной программы. При такой форме работы компьютер не "зависнет" и вам не придется перезагружать процедуру "зануления". Однако, свою исходную программу вам все же придется перезагрузить, так как процедура в машинных кодах внесет непоправимое изменение в номера строк Вашей исходной программы на Бейсике.

Сведущий пользователь без труда сможет внести коррективы в текст исходной программы в машинных кодах с тем, чтобы она избегала нежелательных изменений при

любых вариантах ее использования, однако я не стал этого делать с тем, чтобы программа имела как можно более простой вид и даже начинающий в программировании на ассемблере мог легко в ней разобраться. Тем не менее, если Вы с самого начала будете внимательны и не забудете задать строку с контрольным номером, то Вам не придется надеяться на подстраховку.

Наиболее простым вариантом ввода такой строки было бы набрать:

```
9990 REM ENTER.
```

Однако, как уже упоминалось, Вы можете изменить номер контрольной строки и для этого вам необходимо ввести соответствующие изменения в исходную программу в машинных кодах, в случае, если нобходимый вам номер равен X, то достаточно подать команды:

```
LET a=INT(X/256): LET b=X-256*a:POKE adr+1,b: POKE adr+2,a
```

В данном случае adr - это значение ячейки памяти, начиная с которой вы загрузили процедуру "зануления".

Естественно, что вводить новый номер конечной строки надо перед запуском процедуры.

Для читателей, которые не имеют ни малейшего желания разбираться в программах в машинных кодах, однако желающих использовать данную процедуру, я привожу здесь программу на Бейсике, которая позволит Вам загрузить в память компьютера описанную выше процедуру "зануления" (см. Листинг 2).

Листинг 2.

```
5 REM programming by Michailenko Vadim Mensk 1992
10 CLEAR 63129
20 FOR I=62130 TO 62168
30 READ A: POKE I,A
40 NEXT I
50 REM RANDOMIZE USR 68130
60 DATA 1,6,39,42,83,92,175,119,35,119,35,94,35,86,25,126,254,13,192,35,136,164,40,8,175,
    54,0,35,5,0,24,234,35,126,185,200,43,24,241
```

Данная программа на Бейсике загружает блок машинных кодов, начиная с адреса 62130. Для того, чтобы записать этот блок машинных кодов отдельным файлом, необходимо подать прямую команду:

```
SAVE "имя файла" CODE 62130,39
```

Как уже упоминалось в статье, данная программа рассчитана для работы в произвольной области ОЗУ. Для того, чтобы загрузить ее в необходимую Вам область памяти, следует подать команду:

```
LOAD "имя файла" CODE adr
```

где adr - адрес начала загрузки.

Том 4. Методы защиты программ от копирования.

Введение.

Все, что мы до сих пор рассматривали, по сути было защитой программ от несанкционированного просмотра. Теперь настал момент перейти к более важным моментам - защите от копирования. Поскольку защита от копирования тоже делается программно, то это и есть та самая уязвимая точка, которую и надо защищать от просмотра и мы полагаем, что теперь Вы уже умеете это делать и готовы идти дальше.

Данная книга посвящается вопросам защиты программ от копирования. Этот материал сегодня наиболее интересен для наших программистов, поскольку до сих пор не действует закон об авторском праве на программные продукты. Таким образом, для того, чтобы защитить свои авторские права программисту приходится прибегать к защите от копирования, чтобы только зарегистрированный пользователь мог работать с программой.

Данная проблема сегодня характерна для всех типов компьютеров. "Спектрум" - не исключение. Скорее, даже наоборот и у все большего числа людей возникает стремление

написать свои собственные программы. У них есть для этого и желание и способности но, наступает момент и автор начинает задумывается: а стоит ли это делать это, если ему не удастся защитить свое авторское право на данную разработку.

Естественный выход в подобных ситуациях - ограничение количества неофициальных пользователей. Компьютерное пиратство приобрело в нашей стране в последние годы небывалый размах. Единственным способом хоть как-то сдерживать его оказалось введение защиты компьютерных программ от копирования.

Для "Спектрума" существует несколько различных способов записи программ. В первую очередь, они подразделяются по типу носителей информации: большинство программ записаны на обычных магнитофонных кассетах, в тоже время в последние годы у пользователей все больший интерес приобретает дисковая операционная система.

Мы рассмотрим здесь методы защиты программ, записаных на магнитофонные кассеты. С момента возникновения "Спектрума" данных методов было изобретено очень большое количество, продолжают они совершенствоваться и сегодня. Кроме этого начинают появляться новые методы защиты от копирования.

При использовании приводимого нами материала необходимо учитывать, что большинство программ, которые мы будем рассматривать, написаны в машинных кодах. Это еще раз говорит о том, что высокого уровня профессионализма можно добиться, только изучив программирование на языке Ассемблера. Несмотря на то, что большинство рассматриваемых процедур снабжены подробным комментарием, мы настоятельно рекомендуем Вам перед прочтением данного материала поближе познакомиться с программированием в машинных кодах. Одной из лучших книг, которые нам приходилось встречать по данной тематике, является разработка "ИНФОРКОМа" "Практикум по программированию в машинных кодах".

Мы рекомендуем вам для начала просто ознакомиться с текстом, получить представление об основных принципах и положениях статьи, а потом уже детально изучить проблему по разделам. Мы старались подготовить информацию таким образом, чтобы она не требовала специального запоминания, а усваивалась бы по ходу внимательного прочтения.

Необходимо учитывать, что данный материал тесно связан с той информацией, которая давалась в предыдущих томах. Впрочем, если вы и не читали предыдущих статей, то не отчаивайтесь, этот том отличается определенной автономией и не требует особой подготовки. Он может быть интересен как профессионалам, так и начинающим. Желаем удачи при его изучении!

1. Временные диаграммы и основные характеристики файловой структуры записи на магнитную ленту.

Как Вы уже знаете, роль внешней памяти в вашем компьютере выполняет магнитофонная кассета. Она обеспечивает приемлемую надежность системы при небольшой стоимости носителя. Дополнительным преимуществом является возможность взаимодействия с произвольным, в т.ч. бытовым магнитофоном. Основным недостатком является большое время передачи данных и последовательный доступ к файлам.

Рассмотрим примененную систему кодирования информации на магнитной ленте. Принятое здесь решение дает весьма гибкую систему с достаточной степенью надежности и защищенностью от разброса параметров бытовых магнитофонов.

Основа проста. К порту компьютера с адресом 254 подсоединяется посредством контроллера ULA микрофонное гнездо MIC и выходное EAR. Напряжение в микрофонном гнезде полностью зависит от третьего бита байта, вписываемого в этот порт. Если этот бит равен 0, то выходное напряжение составляет 0,75 Вольта, а если 1 - 1,3 В. Очередная смена установки этого бита на 0 или 1 в компьютерном порту 254 приводит к образованию прямоугольного сигнала, изображенного на диаграмме 1.

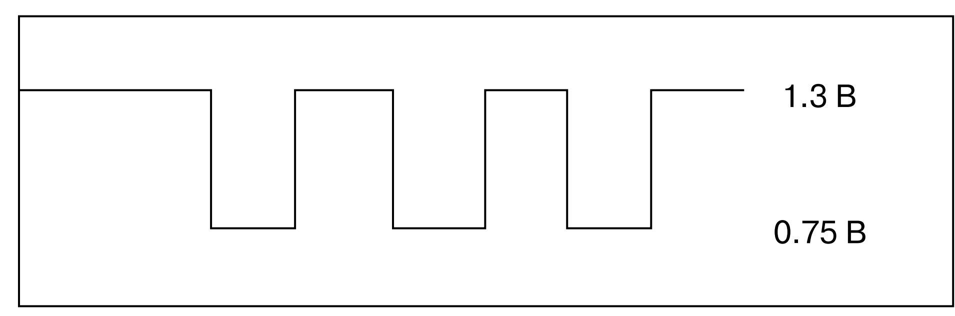


Диаграмма 1. Система кодирования информации на магнитофонной ленте в виде прямоугольных импульсов.

Показанный на диаграмме прямоугольный сигнал можно записать на ленту как звук с определенной частотой, зависящей от времени, в течение которого установка третьего бита есть константа. Рассмотрим это на примере программы, взятой из книги Анджея Кадлофа "Спектрум и магнитофон".

Данная программа попеременно включает и выключает магнитофонный бит компьютерного порта 254:

```
1 OUT 254,0
2 OUT 254,8
3 GOTO 1
```

Подключив усилитель низкой частоты к выходу MIC, мы услышим звук низкого тона. Это объясняется тем, что интерпретатор БЕЙСИКа работает очень медленно. На таком уровне скорости он не позволяет генерировать высокие тона. Поэтому собственные процедуры для записи и считывания с кассеты должны быть написаны только в машинном коде.

Кстати, попутно надо еще и пояснить причину, по которой бордюр телевизионного экрана во время выполнения предыдущей программы становился черным. Три младшие бита байта, выданного на порт 254, определяют действительный цвет бордюра. Это облегчает характерный только для "Спектрума" способ индицирования на экране операций с магнитофоном. Тот, кто пробовал работать с компьютерами других систем, не имеющих такого метода индикации, знает какое это благо.

В машинном коде Z-80 существует обширная группа команд, позволяющая процессору получать данные от внешняя устройств и выдавать данные на эти устройства. Это делается по аналогии с загрузкой данных в регистры микропроцессора из ячеек ОЗУ и выдачей их в ОЗУ на хранение.

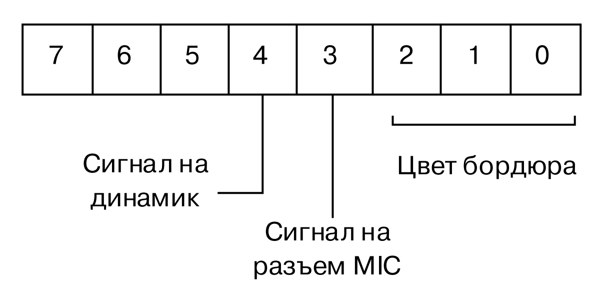
Как Вам уже, вероятно, известно, принцип внутренних и внешних устройств оценивается по отношению к микропроцессору Z-80. Именно поэтому такие части компьютерной системы, как клавиатура, магнитофон и звуковой динамик являются внешними.

Мнемоники команд АССЕМБЛЕРа, отвечающих за ввод/вывод байтов через внешний порт аналогичны операторам БЕЙСИКа IN и OUT. Однако, на языке АССЕМБЛЕРа передаваемые в порт данные рассматриваются как восьмибитные числа, и поскольку порт 254 служит для связи процессора не только с магнитофоном но и с клавиатурой, со звуковым динамиком и с бордюром экрана, у нас могут возникать побочные эффекты, если мы используем порт 254 из БЕЙСИКа.

Рассмотрим раскладку данного порта при записи и чтении информации. При вводе информации в порт 254 шестой бит указывает на наличие сигнала на магнитофонном разъеме (вход в компьютер), причем единице соответствует отсутствие сигнала, в то время как ноль показывает его наличие. Младшие пять битов определяют, какая из пяти клавиш каждого полуряда клавиатуры была нажата. Бит равен нулю, если клавиша была нажата и единице, если нет. Все это наглядно демонстрирует приведенная ниже диаграмма.



Изучим выходную раскладку сигналов компьютерного порта 254. Информация, которая находится в четвертом бите, передает сигнал на звуковой динамик "Спектрума". По третьему биту выдается сигнал на разъем MIC (запись информации на магнитофон). А по младшим трем битам, как мы уже упоминали, выдается сигнал на установку цвета бордюра. Цвет устанавливается в соответствии с номером одного из восьми стандартных цветов "Спектрума". Вся приведенная выше информация наглядно демонстрируется на диаграмме.



Информация, которую мы получаем с магнитофона, не является набором прямоугольных импульсов, а характеризуется сигналом близким к синусоидальному. Однако, такой сигнал тоже поддается компьютерному анализу. На этом принципе основана программа цветомузыки, когда в такт мелодии, игравшей на магнитофоне, экран начинал изменять свой цвет, приведенная в одном из номеров "ZX-РЕВЮ".

Для кодирования данных в "Спектруме" каждый записанный блок состоит из комбинаций четырех различных видов импульсов. Первым генерируется пилотирующий сигнал, при котором смена напряжения наступает регулярно через 619.4 микросекунды, что соответствует 2168 тактам частоты синхронизации микропроцессора Z-80. Генерируемый сигнал имеет частоту 807 Гц. Его продолжительность составляет порядка пяти секунд для заголовка ("хэдера") и около двух секунд для блоков данных. Конец пилотирующего сигнала характеризуется тремя фронтами, образующими так называемый импульс синхронизации (синхроимпульс). Интервалы между ними составляют соответственно 667 и 735 тактов частоты синхронизации микропроцессора. Далее без перерыва во времени пересылаются отдельные биты данных, причем единица представляется двумя фронтами, появляющимися с интервалом 1710 тактов (488.6 микросекунды), в то время как логический ноль представлен 855 тактами (244,3 микросекунды). Интервалы между передаваемыми байтами отсутствуют. Наглядно характеристики сигналов показаны на диаграмме 2.

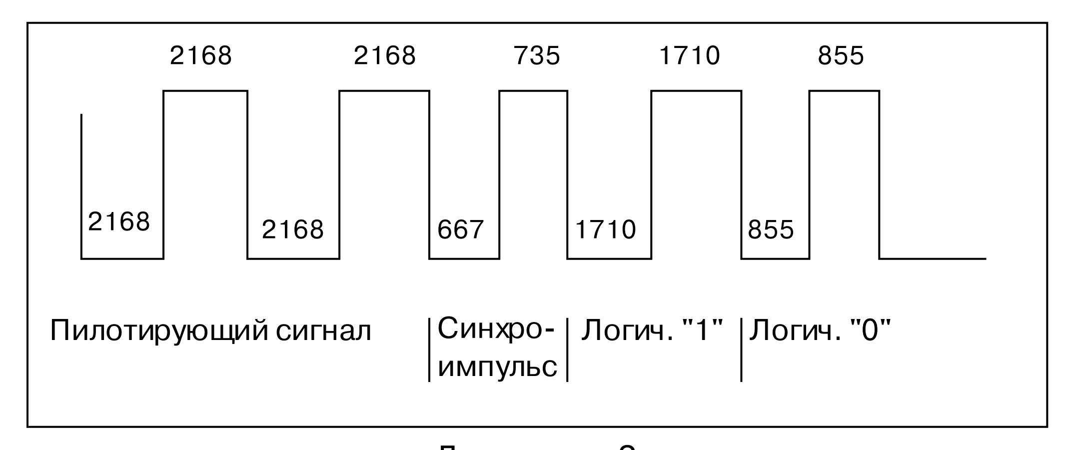


Диаграмма 2

Очень часто защиту программ от копирования основывают на изменении каких-либо стандартных параметров загрузки или записи на магнитофон. Если Вам удалось изменить первичные параметры, то стандартный загрузчик уже не сможет правильно считать вашу программу с магнитофонной ленты. Это сможет сделать лишь специализированная

программа, написанная Вами специально для снятия такого типа защиты (т.е. загрузить эту программу сможет только тот, кто имеет переданный вами нестандартный загрузчик).

Вы можете изменять длительность импульсов пилотирующего сигнала, длительность синхроимпульса, а также длительность логического нуля и логической единицы. Все это приведет к описанному выше эффекту, то есть этим способом Вы сможете защитить свою программу от копирования.

Теоретически ознакомление с вышеописанными временными характеристиками сигналов уже достаточно Вам для написания программ записи и чтения с кассет в формате "Спектрума".

Однако, дело это достаточно сложное и требует очень хорошего знания языка АССЕМБЛЕРа. Еще для правильности интерпретации считываемых файлов необходимо подробно изучить, как производится контроль правильности блока данных. Ведь записывая каждый файл на ленту, "Спектрум" добавляет к нему два байта: один в начале и один в конце. Первый из них сигнализирует о том, является ли данный блок заголовком (значение байта в этом случае равно нулю) или же собственно блоком данных (в этом случае байт принимает значение 255).

Последний байт блока, так называемый байт четности, связан непосредственно с контролем правильности считывания. Его значение записывается в регистр и во время записи последовательного ряда байтов. Принцип контроля основан на применении команды "ИСКЛЮЧАЮЩЕЕ ИЛИ" (XOR). Здесь результат равен единице, если хотя бы один из операндов равен единице, но не оба вместе. В остальных случаях он равен нулю. Перед посылкой каждого восьмибитового байта из регистра L выполняются команды:

```
LD A,H
XOR L
LD H,A
```

В результате этого байт в регистре H содержит информацию о четности появления единицы на данной позиции во всех высланных байтах. Запись его в конце блока обеспечивает возможность контроля. В такте чтения проверяется имеет ли считываемый блок такое же свойство. Если результат последнего байта не совпадает с вычисленным в ходе загрузки значением, выдается сообщение об ошибке загрузки с ленты:

```
TAPE LOADING ERROR
```

Эта система на практике оказывается не только очень простой, но и очень эффективной.

Для того, чтобы научиться защищать собственные программы от копирования, вам необходимо не только представлять, каким образом информация кодируется на магнитной ленте, но и разобраться с основными принципами работы встроенных программ чтения и записи с магнитофона. Эти программы построены так, что они очень просты и непритязательны, если в одной упряжке используются "SAVE"и "LOAD", известна точная длина блока данных и вас не волнует возврат в БЕЙСИК вследствие ошибки или нажатия "BREAK".

Структурная схема заголовка.

Байты	0	1	2..11	12-13	14-15	16-17	-
	Флаг	тип	имя	длина	старт	бейсик-длина	паритет
IX+	-	0	1..10	11-12	13-14	15-16	17

Обычно, при загрузке "Спектрум" полагает что заголовок, говорящий компьютеру как работать, будет получен перед основным блоком и лишь затем последует сам блок. Однако, это характерно лишь для первых программ к данному типу компьютеров. Последние программы в большинстве случаев используют принцип записи блоков данных без заголовка. Это осуществляется при использовании машинных кодов и возможно только тогда, когда точно известны все параметры загружаемого блока. Однако, для того чтобы научиться работать на таком высокопрофессиональном уровне, Вам для начала необходимо

освоить основы, то есть изучить принципы чтения и записи с магнитофона "зашитые" в стандартном ПЗУ "Спектрума".

Для начала рассмотрим структуру заголовка. Его длина составляет 19 байтов, а не семнадцать, как написано в большинстве книг. Тем не менее, только семнадцать байтов должны быть активны, так как процедуры записи и загрузки первый и последний байты определяют сами.

Чтобы согласовать разночтения в литературе относительно длины заголовков давайте считать, что первый и последний байты являются служебными и к пользователю не имеют никакого отношения, а остальные семнадцать - информационными.

Тогда:

Байт 0 - флаговый байт, для заголовков всегда равен нулю, а для блоков данных равен 255.

Байт 1 - содержит число, характеризующее тип записи:

0 -- это БЕЙСИК-программа,

1 -- числовой массив,

2 -- массив символов,

3 -- блок кодов.

Байты 2-11 - содержат имя программы (блока данных).

Байты 12-13 - длина блока. Длина программы кодируется двухбайтным шестнадцатиричным числом. (Для БЕЙСИК-программы это, соответственно, разность между содержимым системных переменных "ELINE - PROG").

Байты 14-15 - хранит в себе начальный адрес загрузки (если это блок машинных кодов) или номер строки автостарта для БЕЙСИК-программ. Если же блок является массивом данных, то для него байт 15 кодируется специальным образом:

биты 0-4 - имя от A=1 до Z=26;

бит 5 - сброшен, если массив - числовой.

бит 6 - активен, если массив - строковый.

бит 7 - активен всегда.

Байт 16-17 - это длина для БЕЙСИКа, то есть разность между содержимым системных переменные "VARS-PROG".

Последний байт - байт четности ("PARITY BYTE"). Он выдается при записи блока на ленту автоматически и при загрузке считывается и проверяется.

Поскольку при загрузке/выгрузке контроль за проходящими байтами ведут через регистр IX микропроцессора (в стандартных процедурах), то для систематизации информации принято отсчитывать эти байты смещением от базы, содержащейся в IX, как показано на структурной схеме заголовка.

Каждый пользователь "Спектрума" обратил внимание на то, что блок информации (файл) на магнитной ленте начинается с синхронизирующего сигнала длительностью две или пять секунд. Частота этого сигнала составляет около восьмисот Герц (период 1.25 миллисекунды). После этого сигнала идет один период специального синхросигнала для которого длительность нуля составляет около 0.19 миллисекунды, а единицы - 0.21 миллисекунды. Затем следуют байты данных передаваемые последовательно, начиная со старшего бита.

Итак, мы с Вами подробно рассмотрели структуру заголовка файла. Структура блока данных фактически ничем не отличается по частоте пилотирующего и длительности импульса синхронизирующего сигнала, однако байт типа принимает значение нуля для заголовков и 255 для блоков данных. Вы можете проверить это, если попытаетесь загрузить два заголовка подряд, то есть загрузить первый заголовок, и перемотав ленту назад, попробовать загрузить его еще раз. У Вас ничего не получится, так как компьютер очень строго следит за типом вводимой информации.

На этом мы пока прервемся, а далее рассмотрим процедуры, которые отвечают за загрузку и выгрузку программ.

(Продолжение следует).

ОШИБКИ ПЗУ

Обзор по материалам зарубежной печати

Сегодня, уважаемые читатели, мы продолжаем разговор о вскрытых ошибках и неточностях стандартного ПЗУ "Спектрума". Начало статьи см. в предыдущем выпуске на стр. 209 - 210.

8. Ошибка CLOSE#.

Профессионалы считают эту ошибку наиболее серьезной из всех. С точки зрения рядового пользователя она, может быть, таковой и не является, поскольку ему редко приходится иметь дело с нестандартными каналами и потоками.

Ошибка проявляется в тех случаях, когда внешняя периферия, имеющая собственное ПЗУ для обслуживания каналов и потоков не подключена. В этой случае, если вы дадите команду на закрывание потока CLOSE #n, а сам поток #n никогда перед этим и не открывался, то Ваш "Спектрум" вместо того, чтобы предупредить Вас о том, что синтаксис неверен, зависает, а иногда (реже) сбрасывается.

Ошибка вызвана тем, что таблица данных, находящаяся в ПЗУ по адресу 1716H (5910 DEC) не заканчивается, как ей положено, нулевым байтом 00.

Интересно отметить, что и фирма "AMSTRAD", перекупив у К.Синклера права на производство "Спектрум"-совместимых машин, не исправила эту ошибку в ПЗУ для "Spectrum+2", хотя другие изменения в ПЗУ сделала. Казалось бы, уж если все равно меняешь ПЗУ (чего не делал сам К. Синклер дабы не снизить совместимость программного обеспечения и не огорчать простых пользователей), то можно было бы и исправить этот дефект.

9. Ошибка CHR$ 9.

Управляющий код CHR$ 9 должен был действовать противоположно коду CHR$ 8. Код CHR$ 8 называется BACKSPACE и вызывет перемещение курсора (текущей позиции печати) влево. Код же CHR$ 9 должен был бы по аналогии называться FORWARDSPACE и вызывать перемещение курсора или позиции печати на одно знакоместо вправо без изменения содержимого текущего знакоместа.

На практике он не делает ни того, ни другого. Более того, в результате его применения, содержимое текущей позиции окрашивается в текущие цвета INK и PAPER, т.е. в нем заложена двойная ошибка.

Листинг 1

```
2A655C     PR_FP_OK    LD HL,(STKEND)      ; HL хранит адрес вершины стека
                                           ; калькулятора.
225F5C                 LD (X_PTR),HL       : Запомнили вершину стека в си-
                                           ; стемной переменной X PTR.
CDE32D                 CALL 2DE3           ; Вызвали процедуру ПЗУ
                                           ; PRINT FP. при этом пятибайт-
                                           ; ное число снялось с вершины
                                           ; стека калькулятора.
2A5F5C                 LD HL,(X_PTR)       ; восстановили старый адрес вер-
                                           ; шины стека к-ра, т.е. теперь
                                           ; HL указывает на новый адрес
                                           ; вершины стека + 5.
11FBFF                 LD DE,FFFB          ; Число FFFBH равно -5 DEC (по
                                           ; правилам двоичной дополнитель-
                                           ; ной арифметики.
19                     ADD HL,DE           ; Теперь HL указывает на новую
                                           ; вершину стека калькулятора.
22655C                 LD (STKEND),HL      ; Запомнили ее в соответствующей
                                           ; системной переменной.
```

```
FD362600                 LD (X_PTR),00       ; Погасили старший байт указателя X PTR.
C9                       RET                 ; Возврат.
```

Первый недосмотр состоит в том, что процедура ПЗУ, выполняющая перемещение курсора вправо (0A3DH = 2621 DEC) должна бы заканчиватъся не командой возврата RET, а командой безусловного перехода JP 0ADC. Вторая же ошибка, связанная с цветом состоит в том, что когда эта процедура работает, надо запоминать состояние системной переменной MASK_T (5C8FH =23695 DEC), затем выставлять в ней число 0FFH, а по окончании работы процедуры восстанавливать запомненное значение.

10. Ошибка CHR$ 8.

Есть ошибки и в процедурах, выполняющих перемещение курсора влево. В большинстве случаев с кодом CHR$ 8 все в порядке, но, к сожалению, не всегда.

Так, если у вас текущей позицией печати является знакоместо с координатами AT 1,0; то BACKSPACE не работает.

Более того, есть возможность смещения влево из координаты 0,0; а это уже совершенная чепуха с неожиданными результатами.

Ошибка находится в процедуре, обслуживающей перемещение курсора влево (0A23H = 2595 DEC). Она проверяет номер экранной строки, на которой установлен курсор, но вместо того, чтобы "отловить" нулевую строку и заблокировать в ней перемещение курсора, делает это для первой строки. Очевидно, в команде программистов у К. Синклера была некоторая несогласованность. Конкретная причина - в том, что по адресу 0A33H = 2611 DEC должно быть число 019H вместо 018H.

11. Ошибка STR$.

Эта ошибка проявляет себя как в БЕЙСИКе, так и при программировании в машинном коде. Вызвать ее очень просто:

```
PRINT "KU-KU" + STR$ 0.5
```

По такой команде компьютер напечатает только 0.5.

Программисты, работающие в машинном коде, могут столкнуться с этой ошибкой при вызове часто встречавшейся процедуры PRINT_FP. Эта процедура находится по адресу 2DE3H = 11747 DEC и служит для того, чтобы выдать на печать по текущему подключенному каналу то действительное число, которое в данный момент находится на вершине стека калькулятора.

Процедура, обрабатывающая оператор STR$ в своей работе тоже обращается к процедуре FRINT_FP и таким образом эта ошибка проникает и в БЕЙСИК.

Эта ошибка происходит в тех случаях, когда число на вершине стека калькулятора находится в интервале от -1 до +1, исключая границы и число 0. Дело в том, что на вершине стека оставляется ошибочный ноль, который и вызывает все проблемы.

Для тех, кто работает на БЕЙСИКе, эту проблему обойти несложно. Достаточно ввести временную переменную, например так:

```
LET a$=STR$ 0.5
PRINT "KU-KU" + a$
```

Для тех, кто работает в машинном коде, в этом случае лучше не пользоваться процедурой PRINT_FP, а заменить ее какой-либо своей, например приведенной в Листинге 1.

12. Ошибки кодов управления цветом.

Если в качестве текущего канала для выдачи информации Вами выбран какой-либо нестандартный канал (иначе говоря, если вы работаете с пользовательским каналом), то операторы управления цветом, например такие, как PAPER 4 дадут сообщение об ошибке

```
C; Nonsense in BASIC.
```

Если подпрограмма, обслуживающая вывод информации в канал, возвращает после своей работы выключенный флаг C (флаг CARRY флагового регистра F).

Чтобы избавиться от ошибки, необходимо предусмотреть, чтобы все процедуры, обслуживающие вывод информации в каналы, возвращались с выключенный флагом CARRY по крайней мере для управляющих кодов от 10 до 15-го, а также для тех параметров,

которые следуют за кодами управления цветом.

С этой ошибкой связана и еще одна, касающаяся кодов управления цветом, но здесь она относится только к операторам временного изменения цвета, т.е. к тем случаям, когда оператор управления цветом является квалификатором оператора PRINT.

Итак, если в качестве текущего установлен канал, отличный от "S" или "K", т.е. последний оператор PRINT печатал не на экран, то команды установки временных цветовых атрибутов в операторе PRINT по ошибке выдадут цветовой код не в текущий, а в предыдущий канал.

Ошибка связана с неспособностью обслуживающей процедуры избрать канал "S" для этой цели по адресу 21E1H = 8673 DEC.

Обойти ошибку можно, если перед каждой командой временной установки цвета вставить пустой оператор PRINT:

```
PRINT;:INK 4
```

И еще одна смежная ошибка, которая относятся только к 128-килобайтным машинам, а точнее говоря - к тому порту RS232, который в них встроен.

На этих моделях команда:

```
LPRINT INK 4
```

вызовет сообщение об ошибке:

```
C; Nonsense in BASIC
```

Этому есть две причины. Во-первых, программа, которая обслуживает вывод на новый канал "p" (встроенный порт RS232) почему-то ошибочно полагает, что за кодом управления цветом должны идти два параметра, а не один. А во-вторых, соответствующая процедура ПЗУ включает флаг CARRY, а к чему это приводит Вы уже знаете.

В идеале на этих моделях по адресу 0086D (для Sp+128) и 0088C (для Sp+2) должен стоять байт 02 вместо 01, и, кроме того, по адресу 0087C (для SP+128) и 0089B (для Sp+2) вместо инструкции

```
CCF
```

должна стоять пара:

```
SCF
CCF
```

13. Ошибка SCREEN$.

Эта ошибка похожа на ошибку STR$. При расчете функции SCREEN$ из БЕЙСИКа на вершине стека калькулятора полученный результат ошибочно дублируется.

В результате такое выражение, как:

```
IF "x" = SCREEN$ (0,0) THEN PRINT "KU-KU"
```

всегда будет печатать "KU-KU", независимо от того, что имеется на экране в позиции с координатами (0,0). К счастью, эта ошибка характерна только для БЕЙСИКа. Соответствующая процедура ПЗУ при вызове ее из машинного кода работает нормально (о том, как ее использовать, мы писали в книге "Элементарная графика", т.1).

В БЕЙСИКе путь обхода этой ошибки прост и выполняется переприсвоением переменной:

```
LET a$ = SCREEN$ (0,0):
IF "x" = a$ THEN PRINT "KU-KU"
```

Ошибки в редакторе

Есть несколько оплошностей, связанных со встроенным редактором "Спектрума".

14. Ошибка Scroll?.

Когда в нижней части экрана при листинге программы появляется сообщение Scroll? или когда там появляется какое-либо сообщение, связанное с обслуживанием магнитофона, то компьютер ждет от вас нажатия какой-либо клавиши.

Проблема состоит в том, что есть несколько комбинации клавиш, которые нажимать при этом нельзя.

Это TRUE VIDEO, INV VIDEO, CAPS LOCK, GRAPHIC и EXTEND MODE.

Если Вы их нажмете, то вместо продолжения работы получите в нижней части экрана последнюю редактированную строку с включенным курсором. Самое интересное, что на 128-килобайтных машинах, работающих в режиме 128K этот курсор будет соответствовать режиму 48K.

Ошибка находится в процедуре, обслуживающей ввод с клавиатуры KEY_INPUT, которая находится в ПЗУ по адресу 10A8H = 4264 DEC. Эта процедура обрабатывает такие комбинации, как CAPS LOCK и пр., но этого не надо делать в данном случае, когда компьютер просто ждет нажатия какой-либо клавиши в ответ на свое сообщение.

15. Ошибка курсора текущей строки.

Эта ошибка проявляется только на 48-килобайтных моделях, поскольку редактор 128-килобайтных машин сильно отличается.

Наберите:

```
9000 PRINT\9001\EDIT
```

(здесь символ "\" обозначает нажатие клавиши "ENTER").

Если при этом у Вас в программе нет строк с номером, большим, чем 9000, то Вы увидите эту ошибку в нижней части экрана, и строке редактирования появится курсор текущей строки (символ ">").

Этой ошибки не было бы, если бы программа OUT_LINE, расположенная в ПЗУ по адресу 1855H = 6229 DEC, проверяла бы четвертый бит системной переменной FLAGS2 и, когда он включен, не печатала бы курсор ">".

16 Ошибка ведущего пробела.

Вы знаете, что операторы и функции БЕЙСИКа в "Спектруме" набираются не по буквам, а словами (токенами). Для того, чтобы отдельные ключевые слова на экране не сливались друг с другом, встроенный редактор автоматически вставляет между ними пробелы, но делает это не очень стабильно, попробуйте, например:

```
CLS: FOR i=1 TO 5: PRINT CHR$ 244: NEXT i
```

В принципе, для того, чтобы компьютер мог решить, когда надо давать ведущий пробел, а когда нет, существует системная переменная FLAGS, когда ведущий пробел не нужен, нулевой бит этой системной переменной должен быть включен.

Проблема была бы решена, если бы этот флаговый бит автоматически включался всякий раз после исполнения команды CLS или при печати управляющих кодов от 00 до 1FH.

17. Ошибка K-режима.

Если при работе с компьютером у Вас включен курсор "K", это означает, что машина находится в командном режиме и следующее нажатие клавиши должно будет восприниматься, как ввод ключевого слова оператора или функции. Например, нажатие на клавишу "p" в этом режиме даст ввод ключевого слова PRINT.

А что будет, если Вы задержите палец на клавише дольше, чем это необходимо? В этой случае, как положено начнется автоповтор, но K-режим останется принудительно включенным.

Одним словом, если Вам надо сделать например:

```
PRINT P
```

или

```
NEXT N
```

то Вы получите вместо этого

```
PRINT PRINT
```

или

```
NEXT NEXT
```

Ошибка находится в процедуре K_REPEAT, расположенной в ПЗУ по адресу 0310H = 784 DEC.

Чтобы ошибки не было, процедура должна вычитать число A5H из кода нажатой клавиши в тех случаях, когда этот код больше, чем E5H. Тогда повторное ключевое слово будет заменено на прописную литеру.

18 Ошибка проверки синтаксиса.

Эта ошибка проявляется только на машинах 48K, а на машинах 128K она мудро игнорируется.

Дело в том, что у Вас есть возможность ввести в программную строку такие ключевые слова, как ERASE, MOVE, FORMAT, CAT, например

ERASE симв. строка

MOVE строка, строка

FORMAT строка

CAT

Очевидно, что эти команды не могут быть выполнены, если у Вас не подключена соответствующая периферия, например INTERFACE ONE с микродрайвом.

И, конечно, при запуске программы на исполнение, она будет прервана с сообщением об ошибке. Спрашивается, почему же нельзя было отловить эту ошибку при проверке синтаксиса перед вводом строки в память?

На 128-килобайтных машинах тоже можно ввести такие ключевые слова, но при запуске программы они будут игнорироваться и восприниматься так, как воспринимается оператор REM.

Ошибки калькулятора

Теперь рассмотрим несколько ошибок, связанных со встроенным в ПЗУ калькулятором. О некоторых из них мы так или иначе уже упоминали в своих прочих работах.

19. Ошибка MOD_DIV.

Эта ошибка связана с работой кода калькулятора 32h. По этой команде со стека калькулятора должны сниматься два верхних пятибайтных числа, например x и y и вместо них на стек должны отправляться

```
x MOD y и
x DIV y
```

(именно в этом порядке).

Напомним, что x MOD y - это остаток от целочисленного деления x на y, а x DIV y - это целая часть частного от деления x на y.

Таким образом,

```
x MOD y = x - y*INT(x/y)
x DIV y = INT (x/y)
```

В своих расчетах процедура, обслуживающая эту функцию калькулятора, использует нулевую ячейку памяти калькулятора M0, а с этой ячейкой есть одна особенность. Дело в том, что при вычислении функции INT эта ячейка коррумпируется, если аргумент при INT меньше нуля. Таким образом, функция MOD_DIV калькулятора дает неверный результат, когда x/y число отрицательное.

Ошибки могло бы и не быть, если бы процедура, занимающаяся расчетом этой функции (а она расположена в ПЗУ по адресу 36A0H = 13964 DEC) использовала бы в своих расчетах не нулевую ячейку памяти калькулятора, а первую (M1).

20. Ошибка E_TO_FP.

В системе команд калькулятора есть команда с кодом 3C. Ее назначение - умножение числа, находящегося на вершине стека калькулятора на множитель, равный 10 в степени A, где A - содержимое аккумулятора микропроцессора.

Вся неприятность в том, что калькулятор после своего включения командой RST 28 не резервирует содержимое аккумулятора, в отличие от содержимого регистра B. Поэтому, к тому времени, как вы воспользуетесь командой калькулятора 3C, есть большая вероятность того, что в аккумуляторе будет не подготовленное вами число, а что-то совсем другое.

Единственный выход - выйти из калькулятора, прогрузить аккумулятор нужным Вам числом, выполнить нужное умножение вызовом процедуры ПЗУ E_TO_FP и снова вернуться в калькулятор:

```
        Endcalc
```

```
LD A, xx
CALL 2D4FH
RST 28
```

Процедура E_TO_FP находится в ПЗУ по адресу 2D4FH = 11599 DEC.

21. Ошибка INKEY$#0.

Обычно нулевой поток представляет собой клавиатуру, поэтому естественно предположить, что INKEY$#0 - то же самое, что и просто INKEY$ без номера потока.

Тем не менее это не так, и почти необратимо INKEY$#0 выдает пустую символьную строку, что делает эту функцию полностью бесполезной.

Надо также заметить, что в системе команд калькулятора есть команда с кодом 1A, которая служит для расчета функции INKEY$#X, где X - число, содержащееся на вершине стека калькулятора. И эта команда калькулятора будет бесполезной, если поток X представляет клавиатуру.

Ошибка находится в подпрограмме ПЗУ по адресу 1634H=5684 DEC, которая устанавливает канал "X" - текущим каналом. В этой подпрограмме по адресу 1638K стоит ошибочная команда RES 5,(FLAGS), выключающая пятый бит системной переменной FLAGS. в результате этого ошибочно отбивается любое нажатие клавиши вместо того, чтобы быть принятый к рассмотрению.

Ошибку можно было бы исправить, если в подпрограмме READ_IN (3645H = 13893 DEC) сохранить значение системной переменной FLAGS на время вызова подпрограммы CHAN_OPEN (1601H = 5633 DEC).

На этом мы заканчиваем обзор ошибок и неточностей в ПЗУ стандартного компьютера "ZX-Spectrum". Конечно же это не все из того, что оттуда можно выудить, но очень экзотические ошибки, которые проявляются например только на машинах типа "ZX-Spectrum+2" и только при подключенном Интерфейсе-1 мы не рассматриваем, поскольку вероятность встретить среди миллионов наших пользователей подобную конфигурацию конечно есть, но она не более сотой доли процента.

Обзор подготовлен по материалам зарубежной печати; основные первоисточники:

1. Dr. Yan Logan, Dr. Frank O'Hara. "The Complete Spectrum ROM Disassembly".
2. Dr. FranK O'Hara "Understanding Your Spectrum".
3. Dr. Yan Logan "Understanding Your Spectrum".
4. Andrew Pennell "Master Your ZX Microdrive".
5. Tony Stratton "Understanding Your Spectrum".
6. Paul Harrison "Understanding Your Spectrum".
7. Stephen Kelly & others "Understanding Your Spectrum".
8. Chris Thornton "Understanding Your Spectrum".

Алексеев А.Г.

ПРОФЕССИОНАЛЬНЫЙ ПОДХОД

Сегодня - продолжение разговора о некоторых приемах, позволяющих придать "профессионализм" вашим программам. Мы поговорим о небольших кодовых блоках, позволяющих воспроизводить различные звуковые эффекты в ваших программах, такие, какие невозножно создать, непосредственно используя оператор "BEEP". Это, например, стрельба из лазерного пистолета, пуск ракеты, торпеды, получение приза и т.д. Также разговор будет о программе "SOUND", позволяющей создавать и редактировать такие звуки. Попутно, в качестве лирического отступления, остановимся еще на одной теме: о том, как усовершенствовать введение числовых параметров при помощи оператора "INPUT". Кроме того, рассматривая готовую программу, мы еще раз проследим все моменты, изложенные в предыдущие статьях: применение блока "ON ERROR GO TO", структура программы, управление программой при помощи меню и т.д.

За основу разработки взята программа "SPECSOUND" фирмы OZ SOFTWARE. Возможно, читателям известен еще один, более ранний прообраз этой программы "DZWIEKI". Взяв за основу принцип формирования кодовых блоков, воспроизводящих звуки, вся программа была изменена настолько, что стала возможна эффективная работа по созданию и редактированию звуков. Кроме того, устранены многие программные ошибки, и сделан перевод на русский язык. В общем, это стала практически другая программа, однако идея осталась той же. Структура самих кодовых блоков, получаемых в результате работы программы, изменена незначительно, лишь настолько, насколько это не повредило совместимости с прототипом программы.

В основе кодового блока одного звука лежит использование подпрограммы из ПЗУ "BEEPER", расположенной по адресу #03B5. При входе в эту подпрограмму, в регистре DE должно быть задано произведение частоты на время звучания звука, (то есть число периодов колебаний), а в регистре HL - величина, эквивалентная значению периода колебаний. Подробнее о работе подпрограммы "BEEPER" было рассказано на страницах ZX-РЕВЮ в разделе "СЕКРЕТЫ ПЗУ" см. N2 за 1991г. СТР. 28.

В общих словах, принцип формирования звука таков. Организуется цикл из последовательного обращения к подпрограмме "BEEPER", но при каждом следующем обращении период колебаний изменяется на определенную величину по сравнению с начальным тоном (может увеличиваться или уменьшаться). При этом, если величина смещения или ступеньки тона, незначительная, а число обращения к подпрограмме "BEEPER" (число ступенек тона) в цикле - большое, то на слух получится плавно изменяющийся звук - "скольжение" тона. Кроме того, получившееся "скольжение" входит в еще один цикл повторений, то есть большое число таких "скольжений" формируют результирующий звук. Изменяя исходные параметры в широких пределах и комбинируя их в различных взаимных сочетаниях, можно получать интересные звуковые эффекты. Имеется также возможность объединения двух или нескольких результирующих звуков в один (последовательное выполнение двух или более звуков), что открывает дополнительные возможности по созданию звуковых эффектов.

Для тех, кто интересуется машинными кодами, рассмотрим подробно блок кодов, воспроизводящий один звук. Если коды Вас не интересуют, то пропустите этот раздел и переходите непосредственно к Бейсик-программе "SOUND".

Блок кодов, воспроизводящий звук.

Длина блока - 32 байта. Его можно загружать в любое место памяти. Стартовый адрес блока равен адресу загрузки. Для примера, расположим его с адреса #AB00, звездочкой справа отмечены параметры, которые могут меняться.

```
AB00 0E00          LD C,#00           (1)
AB02 0605          LD B,#05           (2)  *
AB04 21F401        LD HL,#01F4        (3)  *
```

```
AB07 C5          PUSH BC          (4)
AB08 114600      LD DE,#0046      (5)  *
AB0B E5          PUSH HL          (6)
AB0C CDB503      CALL #03B5       (7)
AB0F E1          POP HL           (8)
AB10 112800      LD DE,#0028      (9)  *
AB13 ED52        SBC HL,DE        (10) *
AB15 C1          POP BC           (11)
AB16 10EF        DJNZ #AB07       (12)
AB18 3E02        LD A,#05         (13) *
AB1A 0C          INC C            (14)
AB1B B9          CP C             (15)
AB1C 20E4        JR NZ,#AB02      (16)
AB1D 00          NOP              (17)
AB1F C9          RET              (18)
```

Сначала (1) обнуляется счетчик числа повторений "скольжения" - (внешнего цикла) - регистр C. Затем (2) в регистр B заносится число ступенек тона (число повторений внутреннего цикла), а в регистр HL - период начального тона (3). Далее начинает выполняться внутренний цикл. На стеке запоминается (4) содержимое регистров B и C (так как оно будет изменено после отработки подпрограммы "BEEPER"). Потом (5) в регистр DE заносится число периодов колебаний, которые будут выработаны подпрограммой "BEEPER". Чем меньше это число, то есть чем меньшее число колебаний будет выполнено перед очередным изменением периода колебаний, тем более плавным на слух будет "скольжение" тона. Другими словами эту величину можно охарактеризовать как дискретность получающегося звука. Затем на стеке запоминается (6) содержимое регистра HL (по той же причине, что и BC), выполняется подпрограмма "BEEPER" (7) и возвращается со стека (8) содержимое HL. теперь надо изменить величину тона перед следующим вызовом подпрограммы "BEEPER". Для этого величина смещения (или ступеньки тона) заносится (9) в регистр DE, после чего (10) отнимается от содержимого регистра HL. (Если вместо SBC задан код команды ADC, тогда DE складывается с HL. Таким образом можно задать повышение или понижение тона "скольжения".) Теперь в HL - новое значение периода колебаний. Далее возвращается со стека (11) значение BC и проверяется величина B - счетчика внутреннего цикла. Если заданное число повторений внутреннего цикла не выполнено, то происходит (12) очередное повторение внутреннего цикла (с адреса #AB07) с измененным значением периода колебаний. А если счетчик внутреннего цикла обнулен, то продолжается выполнение программы. В регистр A заносится (13) число повторений "скольжений" (внешнего цикла). Так как один внешний цикл уже завершен, то происходит увеличение на единицу (14) счетчика внешних циклов - регистра C. Потом это значение сравнивается (15) с тем, что задано в A. Если число в A больше, чем в C (то есть разница не равна нулю), то происходит (16) возврат на повторение внешнего цикла с адреса #AB02 (выполнение следующего "скольжения"). Если заданное число внешних циклов выполнено, то (17), (18) завершение работы - возврат в вызывающую программу. Необходимость команды NOP в конце программы объясняется желанием сохранить совместимость с прообразом программы "SOUND".

Это то, что касается кодового блока. Теперь рассмотрим Бейсик-программу для редактирования кодовых блоков звуков.

Программа "SOUND"

Сначала несколько слов о том, что же может эта программа. Во-первых, она может продемонстрировать 20 готовых звуков, созданных для примера с ее помощью. Прослушав их, Вы сможете примерно оценить "диапазон" возможностей программы. Вы также можете получить для записи на магнитофон любой (или любые несколько, по выбору) или все 20 примеров звуков для дальнейшего использования в своих программах.

Во-вторых, программа позволяет создавать новые звуки и эффективно заниматься редактированием любого из созданных звуков. При создании нового звука можно, взяв за основу один из имеющихся 20 примеров, подробно "рассмотреть" параметры звука и по

мере необходимости изменять их, добиваясь требуемого звучания. Можно также объединить два и более готовых звука в один, чтобы получать комплексные звуковые эффекты. Готовые звуки можно записать на магнитную ленту. Можно также загружать с магнитной ленты для редактирования блоки кодов - звуки, полученные при помощи этой программы.

Область памяти для создания и редактирования звуков определена с адреса 43776 (#AB00) (может быть изменена по Вашему желанию). Максимальное число создаваемых звуков ограничено областью до конца памяти и составляет 680 звуков. Наверное, это заведомо превышает возможности любого пользователя, поэтому в программе не предусмотрена проверка на достижение конца памяти. При необходимости, Вы можете сами организовать такую проверку, если захотите.

При работе, программа отличается "деликатностью", а именно, она предлагает возможные варианты ответа, предупреждает, если результат неудовлетворительный и рекомендует, что надо сделать, чтобы добиться положительного результата.

Рассматривая текст программы, вы увидите, что структура программы - такая же, как была описана в ZX-РЕВЮ N 9-10 за этот год, когда речь шла о дебюте программы "PROG". Поэтому, если вам непонятны некоторые моменты, то еще раз прочтите о дебюте "PROG".

Итак, текст программы "SOUND", затем комментарии.

```
 0 REM КОДЫ
 1 GO TO 100
 3 RANDOMIZE 3: GO SUB 10: GO SUB 7: GO SUB 8: BORDER 1: PAPER 0: INK 6: CLEAR 43775: PRINT
       BRIGHT 1;AT 2,10: "ПРОГРАММА ДЛЯ"; AT 3,6; "РЕДАКТИРОВАНИЯ ЗВУКОВ"; PAPER 2; INK 7;
       DIVERSE 1;AT 8,8; "                  ";AT 9,8: "    S O U N D    "; AT 10,8; "
       "; PAPER 0; INK 2; INVERSE 0;AT 15,3; "OZ SOFTWARE + АЛЕКСЕЕВ А. Г. "
 4 PRINT INK 4; BRIGHT 1;AT 18,12; "01.08.92": BEEP.1,26: BEEP .1,20: GO SUB 20: GO TO 100
 5 SAVE "SOUND" LINE 2
 6 VERIFY "SOUND": GO TO 5
 7 RANDOMIZE PEEK 23635+256*PEEK 23536+773: POKE 23675,PEEK 23670: POKE 23676, PEEK 23671:
       RETURN : REM UDG
 8 RANDOMIZE PEEK 23635+256*PEEK 23536-251: POKE 23606,PEEK 23570: POKE 23607,PEEK 23671:
       RETURN: REM RUS
 9 POKE 23606,0: POKE 23607,60: RETURN : REM LAT
 10 POKE PEEK 23635+256*PEEK 23636+993, PEEK 23670: POKE PEEK 23535+256*PEEK 23636+994, PEEK
       23671: RANDOMIZE USR (PEEK 23635+256*PEEK 23636+941): RETURN
 20 INPUT ; : PRINT #0; PAPER 6; INK 2; BRIGHT 1; AT 1,5:" НАЖМИТЕ ЛЮБУЮ КЛАВИШУ "
 22 RANDOMIZE 24: GO SUB 10: PAUSE 0
 24 IF INKEY$="q" THEN GO TO 9999
 26 RETURN
 30 BORDER 1: PAPER 0: INK 6: CLS
 31 FOR M=1 TO NN: READ M$: PRINT AT (Y0+(M-1)*DY), X0;N$: NEXT M: PRINT INK 4; AT Y1,3;
       "SPACE,DOWN,UP,0,ENTER,BREAK"
 32 IF MM<1 THEN LET MM=NN
 33 IF MM>NN THEN LET MM=1
 34 PRINT AT (Y0+(MM-1)*DY),X0-DX; PAPER 7; INK 2; BRIGHT 1; OVER 2;S$: BEEP .03,2*MM+10
 35 PAUSE 0: LET I=(INKEY$=" " OR INKEY$="6" OR CODE INKEY$=10)+(INKEY$="7" OR CODE
       INKEY$=11)*2+(INKEY$="0" OR CODE INKEY$=12 OR CODE INKEY$=13)*3+(INKEY$="q")*4: GO TO
       (35+I)
 37 PRINT AT (Y0*(MM-1)*DY),X0-DX; OVER 1;S$: LET MM=MM-2*I+3:GO TO 32
 38 BEEP .1,36: RETURN
 39 GO TO 9999
 40 RANDOMIZE 49: GO SUB 10: BORDER 1: PAPER 1: INK 7: CLS
 42 PRINT AT 1,0;"   ЭТА ПРОГРАММА ПРЕДНАЗНАЧЕНА
       ДЛЯ СОЗДАНИЯ БЛОКОВ МАШИННЫХ КО-
       ДОВ, КОТОРЫЕ ВОСПРОИЗВЕДУТ  РАЗ-
       ЛИЧНЫЕ ЗВУКОВЫЕ ЭФФЕКТЫ  В ВАШИХ
       ПРОГРАММАХ  ( НАПРИМЕР, СТРЕЛЬБА
       ИЗ ЛАЗЕРНОГО ПИСТОЛЕТА, ПУСК РА-
       КЕТЫ, ПОЛУЧЕН ИЕ ПРИЗА И ДР. ) "
 44 PRINT '"   БЛОКИ КОДОВ МОГУ Т БЫТЬ РАЗМЕ-
       ЩЕНЫ ЗА RAMTOP ИЛИ ИХ МОЖНО РАС-
```

```
      ПОЛОЖИТЬ В ПЕРВОЙ  (ИЛИ НУЛЕВОЙ)
      СТРОКЕ BASIC ЗА ОПЕРАТОРОМ REM. "
46 PRINT '"    БЛОК КОДОВ, ВОСПРОИЗВОДЯЩИЙ
      ОДИН ЗВУК, ИМЕЕТ ДЛИНУ 32 БАЙТА.
      ЕГО МОЖНО ПЕРЕМЕЩАТЬ  В  ПАМЯТИ.
      СТАРТОВЫЙ АДРЕС ДЛЯ  ВОСПРОИЗВЕ-
      ДЕНИЯ ЗВУКА БУДЕТ  РАВЕН  АДРЕСУ
      ЗАГРУЗКИ. "
48 BEEP .1,20: GO SUB 20: RETURN
50 RANDOMIZE 59: GO SUB 10: PRINT PAPER 6; INK 2; AT 2,6:" ЧИСЛО ЗВУКОВ: "; FLASH 1;B: PAUSE
      50
52 FOR J=1 TO B: PRINT PAPER 6; INK 2: AT 4,9;" ЗВУК НОМЕР ";J;" ": RANDOMIZE USR (U+32*(J-
      1)): PAUSE 60: NEXT J
59 RETURN
60 PRINT PAPER 2: INK 7; BRIGHT 1; FLASH 1;AT 2,6;" ВЫ УВЕРЕНЫ (Y/N)? ": PAUSE 0: RETURN
70 LET SEC=INT (H*C*(2*E+Z«(G-1)*F)*G/8E5): IF SEC<=10 THEN RETURH
72 LET T=INT(SEC/60): LET S=SEC-60*T
74 CLS : PRINT AT 5,0: "   ОБРАЩАЕМ ВАШЕ ВНИМАНИЕ НА ТО,
      ЧТО ВРЕМЯ ЗВУЧАНИЯ ЗВУКА БУДЕТ
      СОСТАВЛЯТЬ ПРИМЕРНО:"; AT 9,9; T: " МИН. ";S;" СЕК. "; AT 11,0;
    "   ВЫ ГОТОВЫ К ПРОСЛУШИВАНИЮ ?"''
    "   ЕСЛИ ДА, ТО НАЖМИТЕ [Y]; ЕСЛИ
      ВОЗВРАТ В МЕНЮ ТО ЛЮБУЮ КЛАВИШУ."''
    "   ПОСЛЕ ВОЗВРАТА  В  МЕНЮ НАДО
      УМЕНЬШИТЬ   ДИСКРЕТНОСТЬ  ЗВУКА.
      ЧИСЛО СТУПЕНЕК ТОНА  ИЛИ  ПЕРИОД
      НАЧАЛЬНОГО ТОНА."
76 BEEP .5,0: PAUSE 0: RETURN
100 LET P=0
110 LET B=20: LET U=PEEK 23635+256*PEEK 23636+1014
190 LET MM=1
200 RANDOMIZE 190: GO SUB 10: RESTORE 200: LET NN=4: LET Y0=6:LET X0=3: LET DY=2: LET DX=1:
      LET S$="                                  ":': LET Y1=18: GO SUB 30: GO TO 200+10*MM
210 LET U=43776: LET E=P: GO TO 300
220 GO SUB 50: GO TO 200
230 GO SUB 6000: LET MM=1: GO TO 200
240 GO SUB 40: LET MM=1: GO TO 200
299 DATA "РАБОТА С РЕДАКТОРОМ ЗВУКОВ","ДЕМОНСТРАЦИЯ ПРИМЕРОВ ЗВУКОВ","ЗАПИСЬ ПРИМЕРОВ
      ЗВУКОВ","ИНФОРМАЦИЯ О ПРОГРАММЕ"
300 RANDOMIZE 390: GO SUB 10: RESTORE 300: LET NN=9: LET Y0=6: LET X0=4: LET DY=1: LET DX=1:
      LET S$="                              ": LET Y1=18: GO SUB 30
302 IF B=0 AND MM<>1 AND MM<>7 AND MM<>9 THEN PRINT PAPER 6;INK 2;AT 2,2:" СНАЧАЛА НАДО
      СОЗДАТЬ ЗВУК ": BEEP .5,0: PAUSE 50: LET MM=1: GO TO 300
304 GO TO 300+10*MM
310 GO TO 1000
320 GO то 2000
330 GO TO 4000
340 IF B<2 THEN PRINT PAPER 6; INK 2: AT 2,1;" СНАЧАЛА НАДО СОЗДАТЬ 2 ЗВУКА ": BEEP .5,0:
      PAUSE 50: LET MM=1: GO TO 300
345 GO TO 5000
350 GO SUB 50: GO TO 300
360 GO SUB 6000: LET MM=1: GO TO 300
370 GO TO 7000
380 LET MM = 1: GO TO 500
390 LET P=B: GO TO 190
399 DATA "СОЗДАНИЕ НОВОГО ЗВУКА","РЕДАКТИРОВАНИЕ ЗВУКА","ИЗМЕНЕНИЕ ЧИСЛА
      ПОВТОРЕНИЙ","СОЕДИНЕНИЕ ДВУХ ЗВУКОВ"," ДЕМОНСТРАЦИЯ ЗВУКОВ", "ЗАПИСЬ ЗВУКОВ",
      "ЗАГРУЗКА ЗВУКОВ", "УДАЛЕНИЕ ЗВУКОВ", "ВОЗВРАТ В ГЛАВНОЕ МЕНЮ"
400 RANDOMIZE 402: GO SUB 10: RESTORE 400: LET NN = 2: LET Y0=9:LET X0=8: LET DY = 2: LET DX
      = 2: LET S$="                   ": LET Y1=18: GO SUB 30
402 GO TO 400+10*MM
410 LET Z=-1: POKE A+20,82: RETURN
420 LET Z=1: POKE A+20,90: RETURN
499 DATA "ПОВЫШЕНИЕ ТОНА","ПОНИЖЕНИЕ ТОНА"
```

```
500 RANDOMIZE 504: GO SUB 10: RESTORE 500: LET NN=2: LET Y0=9: LET X0=3: LET DY=2: LET DX=1:
     LET S$="                                  ": LET Y1=18: GO SUB 30
502 GO SUB 60: IF INKEY$="y" OR INKEY$="Y" THEN GO TO 500+10*MM
504 LET MM=1: GO TO 300 510 LET B=B-1: GO TO 504 520 RUN 3
599 DATA "УДАЛЕНИЕ ПОСЛЕДНЕГО ЗВУКА","  ПЕРЕЗАПУСК  ПРОГРАММЫ"
1000 PAPER 1: INK 7: CLS
1010 PRINT AT 4,0;
     "   ЕСЛИ ВЫ ХОТИТЕ ЗА ОСНОВУ ЗВУ-
     КА ВЗЯТЬ ПРИМЕР, ИЗ ИМЕЮЩИХСЯ  В
     ПРОГРАММЕ, ВВЕДИТЕ НОМЕР ЭТОГО
     ПРИМЕРА." ' ' "  ЕСЛИ НЕТ, ТО НАЖМИТЕ (ENTER)."
1020 LET B=B+1: LET P=B: LET N=B-INT(B/20)*20
1030 RANDOMIZE 1030: GO SUB 10:INPUT ("НОМЕР ПРИМЕРА (1-20): ";N;" : "): LINE F$: IF F$<>""
     THEN LET N=VAL F$
1040 IF N<1 OR N>20 THEN GO TO 1030
1050 PRINT AT 13,0;"   ВАМ В ДИАЛОГЕ БУДУТ ПРЕДЛОЖЕ-
     НЫ ЧИСЛОВЫЕ ПАРАМЕТРЫ. ЕСЛИ ХО-
     ТИТЕ ИЗМЕНИТЬ ИХ,  ВВЕДИТЕ СВОИ
     ЗНАЧЕНИЯ." ' '
     "   ЕСЛИ НЕТ, НАЖИМАЙТЕ [ENTER].
1060 RANDOMIZE 1060: GO SUB 10:LET A=U+32*(B-1): LET Y=PEEK 23635+256*PEEK 23636+1014+32*(N-
     1)
1070 FOR R = 0 TO 31: POKE (A+R),(PEEK (Y+R)): NEXT R
1060 GO SUB 20: GO TO 3000
2000 PAPER 1: INK 7: CLS
2010 RANDOMIZE 2010: GO SUB 10:PRINT AT 21,0: "НОМЕР РЕДАКТИРУЕМОГО ЗВУКА": INPUT ("(1-
     ";B;"):";P;" : "); LINE F$: IF F$<>" THEN LET P=VAL F$
2020 IF P<1 OR P>B THEN GO TO 2010
2030 LET A=U+32*(P-1) 2040 GO TO 3000
2500 CLS : PRINT AT 7,0;"   ИЗМЕНЕНИЕ ТОНА ОЧЕНЬ ВЕЛИКО,
     ПОЭТОМУ В РЕЗУЛЬТАТЕ ТОН СТАНЕТ
     НИЖЕ МИНИНАЛЬНОЙ ВЕЛИЧИНЫ. "
2510 IF Z=-l THEN PRINT AT 9,0;"ВЫШЕ МАКС";
2520 PRINT ''"   НАДО ИЗМЕНИТЬ НАЧАЛЬНЫЙ  ТОН,
     УМЕНЬШИТЬ ВЕЛИЧИНУ СТУПЕНЬКИ ТО-
     НА ИЛИ ЧИСЛО СТУПЕНЕК ТОНА."
2530 BEEP .5,0: GO SUB 20
3000 PAPER 1: INK 7: CLS
3010 LET C=PEEK (A+9)+256*PEEK (A+10): LET E=PEEK (A+5)+256*PEEK (A+6): LET F=PEEK
     (A+17)+256*PEEK(A+18): LET G=PEEK (A+3)
3020 RANDOMIZE 3020: GO SUB 10:CLS : PRINT AT 21,0; "ДИСКРЕТНОСТЬ ТОНА": INPUT ("(1-2000) :
     ";C; " : "); LINE F$: IF F$<>" THEN LET C=VAL F$
3030 IF C<1 OR C>2000 THEN GO TO 3020
3040 POKE A+10, INT (C/256): POKE A+9, INT(C-(PEEK (A+10)*256))
3050 RANDOMIZE 3050: GO SUB 10:CLS : PRINT AT 21,0:"ПЕРИОД НАЧАЛЬНОГО ТОНА": INPUT ("(1-
     20000): ";E;" : "); LINE F$: IF F$<>"" THEN LET E=VAL F$
3060 IF E<1 OR E>20000 THEN GO TO 3050
3070 POKE A+6, INT (E/256): POKE A+5,INT(E-(PEEK (A+6)*256))
3080 IF PEEK (A+20)=82 THEN LET Z=-1: LET MM=1
3090 IF PEEK (A+20)=90 THEN LET Z=1: LET MM=2
3100 GO SUB 100
3110 PAPER 1: INK 7: CLS
3120 RANDOMIZE 3120: GO SUB 10:PRINT AT 21,0;"ВЕЛИЧИНА СТУПЕНЬКИ ТОНА": INPUT ("(1-2000) :
     ";F;" : "); LINE F$: IF F$<>"" THEN LET F=VAL F$
3130 IF F<1 OR F>2000 THEN GO TO 3120
3140 POKE A+18,INT (F/256): POKE A+17,(F-(PEEK(A+18)*256))
3150 RANDOMIZE 3150: GO SUB 10:CLS : PRINT AT 19,0; "ЧИСЛО СТУПЕНЕК ТОНА"''"(ЕСЛИ 1 - ТО
     ПОСТОЯННЫЙ ТОН)": INPUT ("(1-255) : ":G;" : "); LINE F$: IF F$<>"" THEN LET G=VAL F$
3160 IF G<1 OR G>255 THEN GO TO 3150
3170 POKE A+3,G
3180 RANDOMIZE 3000: GO SUB 10: IF E+Z*F*(G-1)<1 OR E+Z*F*(G-1)>20000 THEN GO TO 2500
3190 LET H=1: GO SUB 70: IF SEC>10 AND INKEY$<>"y" AND INKEY$<>"Y" THEN LET MM=2: GO TO 300
3200 CLS : POKE A+24,201: PRINT AT 11,2; "СЛУШАЙТЕ: ТАК ЗВУЧИТ ВАШ ЗВУК": PAUSE 50:
     RANDOMIZE USR A
```

```
3210 PRINT AT 19,0:" НАЖМИТЕ BREAK ДЛЯ ПОВТОРЕНИЯ
     РЕДАКТИРОВАНИЯ ИЛИ ЛЮБУЮ КЛАВИШУ
     ДЛЯ ПРОДОЛЖЕНИЯ...": PAUSE 0
4000 PAPER 1: INK 7: CLS
4010 LET H=PEEK (A+25)-PEEK (A+1)
4020 RANDOMIZE 4020: GO SUB 10:PRINT AT 21,0: "ЧИСЛО ПОВТОРЕНИЙ": INPUT ("(1-255) : ";H;" :
     "); LINE F$: IF F$<>"" THEN LET H=VAL F$ 4030 IF H<1 OR H>255 THEN GO TO 4020
4040 POKE A+1,0: POKE A+24,62: POKE a+25,H
4050 RANDOMIZE 4000: GO SUB 10: GO SUB 70: IF SEC>10 AND INKEY$<>"y" AND INKEY$<>"Y" THEN
     LET MM=3: GO TO 300
4060 RANDOMIZE 4000: GO SUB 10: CLS : PRINT AT 8,11; INVERSE 1; "СЛУШАЙТЕ "; INVERSE 0; AT
     10,7;"ЗВУК НОМЕР ";P;" ГОТОВ": PRINT AT 12,2:"ЗАПУСК - RANDOMIZE USR ";A: PAUSE 50:
     RANDOMIZE USR A
4070 PRINT AT 19,0:"   НАЖМИТЕ BREAK  ДЛЯ  ИЗМЕНЕНИЯ
     ЧИСЛА ПОВТОРЕНИЙ  ИЛИ ЛЮБУЮ КЛА-
     ВИШУ ДЛЯ ПРОДОЛЖЕНИЯ...": PAUSE 0
4080 LET MM=2: GO TO 300 5000 PAPER 1: INK 7: CLS : LET D=1: IF B=2 THEN GO TO 5100 5010
     PRINT AT 10,0;"   СОЕДИНИТЬ МОЖНО ДВА СОСЕДНИХ
     ЗВУКА. ДЛЯ ЭТОГО ВВЕДИТЕ НОМЕР
     ПЕРВОГО ИЗ ДВУХ СМЕЖНЫХ ЗВУКОВ,
     КОТОРЫЕ НАДО СОЕДИНИТЬ."
5020 RANDOMIZE 5020: GO SUB 10: INPUT ("НОМЕР ЗВУКА (1-";B-1;") : ";D;" : "); LINE F$: IF
     F$<>"" THEN LET D=VAL F$
5030 IF D<1 OR D>B-1 THEN GO TO 5020
5100 LET A=U+32*(D-1): POKE A+31,0
5110 RANDOMIZE 300: GO SUB 10: CLS : PRINT INVERSE 1;AT 7,10;"СЛУШАЙТЕ "; INVERSE 0''" ЗВУК
     НОМЕР ";D;" + ЗВУК НОМЕР ";D+1''"ЗАПУСК - RANDOMIZE USR ";A; PAUSE 50: RANDOMIZE USR
     A
5120 RANDOMIZE 5200: GO SUB 10:PRINT AT 19,0;"   НАЖМИТЕ BREAK ДЛЯ РАЗДЕЛЕНИЯ
     ЗВУКОВ ИЛИ ЛЮБУЮ КЛАВИШУ  ДЛЯ
     ПРОДОЛЖЕНИЯ...":PAUSE 0
5130 LET MM=2: GO TO 300
5200 POKE A+31,201: LET MM=4: GO TO 300
6000 PAPER 1: INK 7: CLS 6010 LET O=1: IF B=1 THEN LET I= 32: GO TO 6100
6020 PRINT AT 8,0;"    СНАЧАЛА ВВЕДИТЕ НОМЕР  ЗВУКА,
     С КОТОРОГО ХОТИТЕ НАЧАТЬ ЗАПИСЬ,
     ЗАТЕМ НОМЕР ЗВУКА, КОТОРЫМ ХОТИ-
     ТЕ ЗАВЕРШИТЬ ЗАПИСЬ. "
6030 RANDOMIZE 6030: GO SUB 10: INPUT ("НАЧАЛЬНЫЙ ЗВУК (1-";B;" ) : ";O; " : "); LINE F$: IF
     F$<> "" THEN LET O=VAL F$
6040 IF O<1 OR O>B THEN GO TO 6030
6050 LET V=B: IF O=B THEN LET I=32: GO TO 6100
6060 RANDOMIZE 6060: GO SUB 10: INPUT ("КОНЕЧНЫЙ ЗВУК (";O;"-";B;") : ";V;" : "); LINE F$:
     IF F$<>"" THEN LET V=VAL F$
6070 IF V<O OR V>B THEN GO TO 6060
6080 LET I=(V-O+1)*32
6090 IF PEEK (U+(O-I)*32+(I-1))=0 THEN LET I = I+32: GO TO 6090
6100. RANDOMIZE 6200: GO SUB 10: CLS : PRINT AT 21,0; "ИМЯ ФАЙЛА : ": GO SUB 9: INPUT F$
6110 CLS : SAVE F$CODE U+32*(O-I),I
6120 GO SUB 8: CLS : PRINT AT 1,0; "   КОГДА  ВЫ  БУДЕТЕ  ЗАПИСАННЫЙ
     БЛОК  КОДОВ  ВСТРАИВАТЬ  В  ПРО-
     ГРАММУ, ПОМНИТЕ, ЧТО ЕСЛИ ВЫ
     РАСПОЛАГАЕТЕ ЕГО В ПЕРВОЙ (НУЛЕ-
     ВОЙ) СТРОКЕ BASIC, ТО ВЫ ДОЛЖНЫ
     В ЭТОЙ СТРОКЕ ПОСЛЕ ОПЕРАТОРА
     REM НАБРАТЬ "; FLASH 1;I; FLASH 0; " ПРОБЕЛОВ. "
6130 PRINT '"   ЗАГРУЗИТЕ ЗАПИСАННЫИ БЛОК КО-
     ДОВ: LOAD ""ИМЯ"" CODE 23760"'"   АДРЕСА ЗАПУСКА  ЗВУКОВ  БУДУТ
     РАВНЫ: 23760+32* (N-1 ), ГДЕ N -
     НОМЕР ЗВУКА ИЗ ЗАПИСАННОГО БЛОКА
     ЗВУКОВ."''"   ЕСЛИ ВЫ РАСПОЛАГАЕТЕ КОДЫ ЗА
     RAMTOP, Т О: LOAD "ИМЯ" CODE ADR
     ГДЕ ADR >=RAMTOP+1. СТАРТОВЫЕ АД-
     РЕСА ЗВУКОВ БУДУТ ADR+32*(N-1)."
6140 BEEP .1,20: GO SUB 20
```

```
6200 GO SUB 6: RETURN
7000 RANDOMIZE 7100: GO SUB 10: GO SUB 9: PAPER 1: INK 7: CLS
7010 LOAD ""CODE (U+B*32)
7020 LET LEN = PEEK (PEEK 23649+256*PEEK 23650+28)+256*PEEK (PEEK 23649+256*PEEK 23650+29)
7030 LET P=B+1: LET B=B+INT (LEN/32)
7040 GO SUB 8: PRINT ' ' ' "ЧИСЛО ЗАГРУЖЕННЫХ ЗВУКОВ: "; INT (LEN/32)
7050 BEEP .1,20: GO SUB 20
7060 LET MM=2: GO TO 300
7100 GO SUB 8: PRINT ' ' 'TAB 4; FLASH 1; "   ОШИБКА  МАГНИТОФОНА  " 'FLASH 0' ' ' "
      ПОВТОРИТЬ ЧТЕНИЕ- [Y]," ' ' " ВОЗВРАТ В МЕНЮ - ЛЮБАЯ КЛАВИША"
7110 BEEP .5.0: GO SUB 20
7120 IF INKEY$="Y" THEN GO TO 7000
7130 LET MM=7: GO TO 300
9999 PAPER 1: INK 7
```

Переменные, используемые в программе.
P - текущий номер звука.
B - суммарное число звуков в блоке кодов.
U - начальный адрес массива кодов, отведенного под звуки.
N - номер звука из примеров, который берется за основу при создании новых звуков.
F$ - стринг для оператора INPUT, значение которого присваивается переменным.
A - начальный адрес звука, являвшегося текущим.
Y - начальный адрес выбранного примера звука.
R - счетчик перебрасываемых байтов при создании основы звука.
C - дискретность тона.
E - период начального тона.
F - величина ступеньки тона.
Z - признак увеличения или уменьшения тона.
G - число ступенек тона.
J - номер демонстрируемого звука.
H - число повторений "скольжений" тона.
SEC - примерное время звучания звука в секундах.
T, S - то же, но соответственно в минутах и секундах.
D - номер первого из двух смежных звуков, которые надо соединить.
O - начальный звук для записи.
V - конечный звук для записи.
I - длина блока кодов для записи на магнитную ленту.
LEN - длина блока кодов при загрузке звуков с ленты.

О нулевой строке и о кодах, размещенных там - чуть позже. А сейчас об остальных строках программы.

Автостарт программы - со строки 2. Так как для работы программы не нужны никакие кодовые блоки, кроме расположенных в нулевой строке, то строка 2 в программе отсутствует. Строка 3 - вывод на экран заставки. (При этом я считаю "делом чести" для себя указывать в заставке фирму, создавшую прототип этой программы.) После вывода заставки, строка 4 адресует на непосредственное начало программы - строку 100.

В строках 100, 110 устанавливаются начальные параметры. Обнуляется счетчик текущего номера звука (P), максимальное число звуков в блоке кодов (B) устанавливается равным 20 - именно столько готовых примеров находится в программе. Определяется начальный адрес (U) области кодов примеров звуков (он может измениться в случае работы с дисковой операционной системой TR-DOS или при подключении ИНТЕРФЕЙСА-1).

Строка 190 определяет начальное положение указателя главного меню на первый пункт. Затем со строки 200 запускается главное меню.

В главном меню 4 пункта:

```
[ РАБОТА С РЕДАКТОРОМ ЗВУКОВ ]
  ДЕМОНСТРАЦИЯ ПРИМЕРОВ ЗВУКОВ
  ЗАПИСЬ ПРИМЕРОВ ЗВУКОВ
  ИНФОРМАЦИЯ О ПРОГРАММЕ
```

Первый пункт - выделен, так как он наиболее вероятен для выбора. Работа меню подробно была изложена в ZX-РЕВЮ N 9-10 за этот год см. стр. 197. Там же подробно описаны переменные, касающиеся работы меню, поэтому эту часть - пропускаем. Отмечу только, что несколько изменена логика работы блока "ON ERROR GO TO", касающаяся переходов из одного меню в другое, но на этом мы еще остановимся. А также добавлен опрос клавиши "Q" и переход, в случае ее нажатия, на строку 9999, аналогично тому, как это делается в строке 24 для остановки программы.

При выборе первого пункта, строка 210 запускает второе меню. При этом принимает заданное значение начальный адрес области памяти, определенной под создаваемые звуки (U), а максимальное число созданных звуков (B) становится равным P, то есть пока что - нулю.

При выборе второго пункта выполняется (строка 220) подпрограмма GO SUB 50 - это демонстрация готовых звуков, затем опять переход в главное меню. Перед демонстрацией Вам показывается, сколько созданных звуков находится в памяти. Если в этот момент нажать "BREAK", то произойдет завершение подпрограммы путем перехода на строку 59. При помощи "BREAK" можно прервать подпрограмму демонстрации звуков в любом ее месте, но только если клавиша удерживается нажатой до завершения очередного звука, то есть до окончания программы в машинных кодах и возврата в Бейсик, когда возобновит свое действие блок "ON ERROR GO TO".

При выборе третьего пункта выполняется большая подпрограмма GO SUB 6000 - это запись звуков на магнитную ленту, после чего происходит возврат в главвое меню с выделением первого пункта.

При выборе четвертого пункта на экран выводится некоторая вступительная информация о программе (при помощи GO SUB 40).

Вернемся к действиям, происходящим в результате выбора первого пункта главного меню. Строка 210 адресует программу на строку 300 - это запуск второго меню с выделенным первым пунктом. (При этом нет необходимости задавать, как положено, LET MM=1: GO TO 300, так как MM и так равно 1.)

Второе меню выглядит следующим образом:

```
[ СОЗДАНИЕ НОВОГО ЗВУКА ]
  РЕДАКТИРОВАНИЕ ЗВУКА
  ИЗМЕНЕНИЕ ЧИСЛА ПОВТОРЕНИЙ
  СОЕДИНЕНИЕ ДВУХ ЗВУКОВ
  ДЕМОНСТРАЦИЯ ЗВУКОВ
  ЗАПИСЬ ЗВУКОВ
  ЗАГРУЗКА ЗВУКОВ
  УДАЛЕНИЕ ЗВУКОВ
  ВОЗВРАТ В ГЛАВНОЕ МЕНЮ
```

При работе второго меню после выполнения подпрограммы меню GO SUB 30 идет строка 302, в которой проверяется допустимость тех или иных действии. Так, если еще не создано ни одного звука, то не может идти речи о редактировании или демонстрации звуков или записи звуков и т.п. В этом случае на экран выводится табличка "СНАЧАЛА НАДО СОЗДАТЬ ЗВУК", затем предупредительный звуковой сигнал, после чего возобновляется работа второго меню.

А теперь небольшое лирическое отступление, необходимое для пояснения дальнейших строк программы. Кстати Вы можете использовать изложенный прием и в других своих программах.

Ввод параметров при помощи оператора INPUT.

Для того, чтобы сформировать звук, надо в диалоге задать несколько исходных

параметров: дискретность тона, период начального тона, величина ступеньки тона, число ступенек тона, число повторений. Для человека, первый раз запустившего программу, эти параметры - "китайская грамота". Что такое, например, дискретность тона? Даже если указать пределы, в которых может изменяться вводимый параметр, например так:

```
INPUT "ДИСКРЕТНОСТЬ ТОНА (1-2000) "; C
```

то это еще немногим поможет, так как все параметры взаимоувязаны между собой и, скорее всего, величина, взятая наобум из указанного диапазона, приведет к тому, что звука не получится, так как результирующие параметры выйдут за допустимые пределы. Чтобы не блуждать в темноте с завязанными глазами, программа должна подсказывать оператору возможные варианты ответа - такие, что если все время соглашаться с этими ответами (нажимая ENTER), то в конце получится что-нибудь осмысленное. Так должно быть в идеале.

Реализуются эти требования следующим образом. Например, если дискретность тона обозначить C, то процедура ввода может выглядеть так (номера строк даны условно):

```
3010 LET C=5
3020 PRINT AT 21,0;"ДИСКРЕТНОСТЬ ТОНА": INPUT ("(1-2000) : ";C; " : "); LINE F$: IF F$<>""
     THEN LET C=VAL F$
3030 IF C<1 OR C>2000 THEN GO TO 3020
```

При этом в строке 3010 или где-либо в другом месте, например, в начале программы, указывается то значение, которое будет принято программой по соглашению (если Вы просто нажмете ENTER, ничего не вводя). В строке 3020 на экран выводится наименование параметра, требующего ввода, далее идет информация о допустимых пределах и затем предлагается вариант для ввода. Если согласен с ответом - нажимай ENTER, если нет - то задай свое значение и нажми ENTER. Для реализации этого способа потребовалось вместо числового значения C вводить стринг F$, а чтобы при вводе не появлялись кавычки, как обычно при вводе стринга, то вместо INPUT F$ используется INPUT LINE F$.

Строка 3030 проверяет вводимый параметр на допустимые пределы, например, набирая "1000", у Вас случайно набрался лишний ноль, но Вы этого не заметили и уверены в правильности ввода. Если программа просто откажется принять введенное число, то это вызовет у Вас недоумение: "Как же так, ведь было сказано ввести число от 1 до 2000, я и ввел 1000, а что-то программа отказывается работать". В таком случае строка 3030 вернет программу на строку 3020, где Вам будет "показано" введенное Вами ошибочное число. Теперь Вы имеете возможность исправить ошибку, повторив ввод.

Заканчивая лирическое отступление отмечу, что на протяжении всей программы ввод параметров осуществляется такими процедурами. А теперь возвращаемся к работе программы, когда на экране второе меню.

При выборе первого пункта меню, строка 310 переводит программу на строку 1000. С этой строки выполняются действия по созданию в памяти основы нового звука. Строка 1020 увеличивает на 1 число созданных звуков (B), затем этот новый звук становится текущим (P=B), затем подготавливается значение (N) для оператора INPUT в следующей строке. N - это номер примера из набора демонстрационных звуков. Если Вы хотите выбрать какой-то конкретный понравившийся пример, то введите его номер, если нет, то нажмите ENTER, при этом будет взят один из примеров. Обратите внимание на активизацию блока "ON ERROR GO TO" при работе INPUT. Конструкция:

```
RANDOMIZE 1030: GO SUB 10
```

в строке 1030 вернет программу на эту строку в случае ошибки при вводе. Строка 1040 также вернет программу для повторного ввода на строку 1030 в том случае, если вводимая величина выходит за допустимые пределы.

После вывода на экран комментария, строки 1060 и 1070 произведут переброску блока кодов длиной 32 байта из области памяти примеров звуков в область памяти для создания и редактирования звуков. Конструкция:

```
RANDOMIZE 1060: GO SUB 10
```

"подстраховывает" процесс создания в памяти основы звука. Так как этот процесс занимает порядка 1 секунды, то можно успеть в это время нажать "BREAK". В этом случае процесс создания основы звука будет повторен сначала и работа программы не пострадает. Строка 1080 передает управление на строку 3000 - это основная часть по редактированию

звука. О ней - через один абзац.

В том случае, если находясь во втором меню, Вы выберете 2 пункт, то строка 320 передаст управление на строку 2000 - эта часть программы позволяет задать номер звука для редактирования из создаваемого блока звуков. По соглашению, редактируемым является последний созданный Вами звук, но Вы можете выбрать любой из тех, которые создали. Работа оператора INPUT аналогична рассмотренной выше. Строка 2030 определяет начальный адрес текущего звука. Строка 2040 передает управление на основную часть программы по редактированию звука - строку 3000.

В строке 3010 происходит присвоение параметров выбранного примера звука числовым переменным для дальнейшего использования их в качестве "предлагаемых по соглашению" параметров при работе оператора INPUT. Далее идет несколько запросов параметров. Строки 3020, 3030 - ввод "дискретности звука". Строка 3040 - занесение полученных значений в память. Строки 3050-3070 аналогично ввод "периода начального тона". Далее в строках 3060, 3090 происходит подготовка параметров для работы следующего, вспомогательного меню со строки 400, которое вызывается как подпрограмма, чтобы можно было не беспокоиться о номере строки для возврата из этого меню в программу редактирования звука. Это меню задает параметр Z - повышение тона или понижение тона. Если задано повышение тона (то есть уменьшение периода колебаний), то Z=-1 и в память вводится число 82, так как 237, 82 - это коды команды SBC - вычитание. Если задано понижение тона (то есть увеличение периода колебаний), то Z=1 и в память вводится число 90, так как 237, 90 - это коды команды ADC - сложение. После возврата по RETURN в строке 410 или 420 - продолжение программы со строки 3110.

Строки 3120-3140 - ввод значения величины "ступеньки тона". Строки 3150-3170 - ввод "числа ступенек тона".

Далее можно подвести некоторые предварительные итоги, проверив взаимоувязку параметров между собой. Для этого в строке 3180 подсчитывается величина тона, получающаяся при завершении звука, то есть на "последней ступеньке" тона. Изменение тона не должно приводить к тому, что тон выйдет за допустимые пределы. Если это все же происходит, то программа возвращается назад, на строку 2500. Здесь выводится сообщение о недопустимом изменении тона и рекомендуется, что надо сделать для корректировки. После предупредительного звукового сигнала и паузы в строке 2530, переходим опять на начало редактирования звука, на строку 3000. В процессе нового ввода Вам будут показаны введенные Вами значения и Вы сможете скорректировать их согласно указаниям строк 2500 - 2520.

Если проверка в строке 3180 прошла успешно, то программа переходит к следующей проверке в строке 3190. Здесь производится приблизительный подсчет времени звучания одного (H=1) "скольжения" тона. Для этого вызывается подпрограмма со строки 70. Если время звучания не превышает 10 секунд, то - возврат в программу, ничего не делая. Если превышает 10 секунд, то в строке 72 происходит перевод этого времени в минуты и секунды для вывода предупреждения. Оно выводится строкой 74. При этом даются рекомендации, что надо сделать, чтобы уменьшить время звучания. Однако, может быть, Вам специально нужно сделать звук такой длины. Поэтому Вы можете продолжить работу программы, дав согласие на демонстрацию звука, нажав "Y". В противном случае строка 3190 вернет программу во второе меню. Если вместо "Y" нажать "BREAK", то сразу окажемся на строке 3000 - то есть без возврата в меню начинается повторное редактирование. Это происходит благодаря установке в строке 3160 блока "ON ERROR GO TO" на строку 3000.

Теперь, когда пройдены все проверки, в строке 3200 Вы можете услышать результаты своего труда. Здесь Вам демонстрируется одно "скольжение" тона. Если Вам результаты сразу не нравятся, Вы можете нажать "BREAK" и, минуя меню, быстро вернуться на строку 3000 для повторного редактирования. (Как Вы помните, блок "ON ERROR GO TO" настраивается на эту строку в строке 3160). Если результаты вас устраивают, то продолжается редактирование звука со строки 4000.

Строка 4000. Сюда мы попадем, также, если во втором меню выбрать третий пункт (см. строку 330). Эта часть программы задает число повторений "скольжений". Строки 4010-

4040 - ввод значения в память - аналогично тому, как это делалось раньше. Далее, в строке 4050 еще раз производится проверка времени звучания заданного числа "скольжений" тона. И если оно больше 10 секунд, то, как и раньше, требуется подтверждение "Y" для демонстрации звука, иначе - возврат во второе меню. При этом, если вместо "Y" Вы нажмете "BREAK", то сразу же попадете на строку 4020 согласно последней установке блока "ON ERROR GO TO", которое было сделано в строке 4020. Если была нажата клавиша "Y", то в строке 4050 демонстрируется готовый звук, после чего процесс редактирования заканчивается и строка 4060 возвращает программу во второе меню.

Если во втором меню выбран четвертый пункт, то есть соединение двух звуков в один, то необходимо сначала убедиться в том, что эти два звука у нас имеются в наличии. Эта проверка выполняется в строке 340. Если число созданных звуков больше или равно двум, то разрешается дальнейшая работа, где строка 345 переводит программу на строку 5000, иначе - вывод предупредительной таблички со звуковым сигналом и возврат во второе меню.

Программа со строки 5000 - это соединение двух звуков. Соединение производится путем замены команды RET в конце кодового блока первого звука, командой NOP, в результате чего после завершения первого звука не произойдет возврат в Бейсик, а начнется выполнение следующего по порядку звука.

Естественно, что первый звук можно соединить только со вторым, второй - только с третьим и т.д. Если создано всего два звука (B=2), то других вариантов, кроме как соединить первый со вторым - не может быть и программа со строки 5000 переходит на строку 5100, а если звуков несколько, то следует запрос: надо указать номер первого из двух смежных звуков, которые надо соединить. Ввод этого значения - в строках 5010 -5030. После того, как в строке 5100 произойдет непосредственно замена команды RET командой NOP, вам демонстрируется получившийся звук. Если он Вам не понравится, Вы можете опять разъединить звуки, если в момент паузы в строке 5120, после вывода на экран комментария, нажать "BREAK". При нажатии "BREAK" программа переходит на строку 5200, где команда NOP опять заменяется командой RET. В завершение этой части программы запускается второе меню.

Если во втором меню выбрать пятый пункт - демонстрация звуков, то в строке 350, как и в первом меню (см. строку 220), вызывается подпрограмма демонстрации звуков GO SUB 50, после чего опять запуск второго меню. Этим режимом можно также воспользоваться, если Вы просто хотите посмотреть, сколько звуков уже создано. Для этого, как только на экране появится: "ЧИСЛО ЗВУКОВ:", прервите выполнение этой подпрограммы, нажав "BREAK".

При выборе шестого пункта второго меню - записи звуков - выполняется большая подпрограмма записи GO SUB 6000, аналогично тому, как она выполнялись из первого меню. Отмечу только один момент. Например у вас создано три звука. При этом Вы соединили между собой второй и третий звуки, заменив команду RET в конце второго звука командой NOP (при этом второй звук стал вдвое большей длины). Теперь у Вас имеется два звука: первый - обычной длины, и второй - двойной длины. Это необходимо учесть при записи звуков. Так, если Вы хотите записать результаты работы на магнитофон, то в ответ на запросы о номере начального и конечного звуков для записи, Вы логично ответите: 1 и 2. При этом программа сама должна разобраться в том, что в конце блока кодов длиной 32*2=64 байта отсутствует RET. Иначе записанный блок кодов окажется неработоспособным. То есть, если в конце отсутствует RET, то длина кодового блока для записи увеличивается на 32 байта, после чего опять должна быть выполнена проверка на наличие RET в конце блока. Эта проверка выполняется в строке 6090, после чего следует запись (строки с 6100). После записи на экран выводятся краткие комментарии по дальнейшему использованию звуков в Ваших программах. После завершения подпрограммы записи - возврат туда, откуда подпрограмма была вызвана - в первое или второе меню.

При выборе седьмого пункта второго меню - загрузки звуков, программа в строке 370 переводится на строку 7000. Перед началом загрузки блок "ON ERROR GO TO" в строке 7000

настраивается на переход на строку 7100 в случае ошибки магнитофона. Далее идет загрузка блока кодов с ленты в область памяти, непосредственно за последним созданным звуком (строка 7010). Далее определяется длина загруженного блока кодов (строка 7020). Текущим устанавливается первый звук из загруженного блока звуков, а общее число звуков увеличивается на число загруженных звуков (строка 7030), которое индицируется на экране (строка 7040). Далее - вызов второго меню. В случае ошибки магнитофона или нажатия "BREAK" при ожидании загрузки программа переходит на строку 7100. Можно возобновить загрузку, нажав "Y", иначе - переход во второе меню.

В том случае, если во втором меню будет выбран восьмой пункт - удаление звуков, то строка 360 вызывает новое меню, которое расположено со строки 500. Здесь два варианта: удаление последнего звука или вообще удаление всех звуков путем перезапуска программы. При этом в обоих случаях требуется подтверждение для удаления звуков, иначе строка 504 возвратит программу во второе меню. Строка 510 - уменьшение на 1 числа созданных звуков, строка 520 - перезапуск всей программы аналогично перезагрузке с ленты.

И, наконец, последний пункт второго меню - возврат в главное меню. Вообще говоря, этот пункт меню лишний, возврат в главное меню производится точно так же при помощи нажатия на клавишу "BREAK", однако помня об идеологии меню, то есть если Вы активно пользуетесь джойстиком, то этот пункт меню может придать большее удобство при работе.

Для того, чтобы реализовать возможность перехода из второго меню в главное меню, в программе "SOUND" применима несколько иная логика работы блока "ON ERROR GO TO", по сравнению с программой "PRIM". Там при нажатии "BREAK", находясь в первом или втором меню, можно было перезапустить программу сначала с выводом титульной заставки (аналогично перезагрузке с ленты), для чего в начале строки 30 - в начале подпрограммы меню стояла конструкция, активизирующая переход при ошибке (нажатий "BREAK") на строку 3. В программе "SOUND" такое нажатие "BREAK" (случайное, конечно) было бы крайне опасно, так как оно может в один миг уничтожить все созданные Вами звуки, то есть все результаты труда. Поэтому, когда на экране какое-нибудь меню, программа должна по разному реагировать на нажатие "BREAK" в зависимости от ситуации.

Рассмотрим подробнее реакцию программы в различных меню на нажатие "BREAK". Прежде всего, теперь в подпрограмме меню GO SUB 30 Вы не видите конструкции, активизирующей блок "ON ERROR GO TO". Эта активизация должна быть выполнена при входе в соответствующее меню до начала выполнения подпрограммы GO SUB 30. Это происходит в строках 200, 300 и т.д., то есть в начальных строках каждого меню. А теперь конкретно о каждом меню.

Первое меню. Строка 200. Конструкция RANDOMIZE 190: GO SUB 10 вызывает при нажатии "BREAK" опять же это первое меню, переходом на строку 190. То есть, нажимая "BREAK", невозможно никуда попасть дальше первого меню. Так предохраняются от уничтожения результаты редактирования. Кроме этого, появляется небольшое дополнительное удобство на практике - быстрый переход на первый пункт меню из середины меню: вместо многократного или длительного нажатия "ВВЕРХ" или "ВНИЗ" достаточно один раз нажать "BREAK".

Второе меню. Строка 300. При нажатии "BREAK" произойдет переход на строку 390, то есть возврат в первое меню.

Меню "повышение - понижение тона". Строка 400. Здесь при нажатии "BREAK" будет переход на строку 402, то есть та же реакция, что и на нажатие ENTER, то есть принятие параметра по соглашению.

Меню удаления звуков. Строка 500. Здесь при нажатии "BREAK", как и любой другой клавиши, происходит переход на строку 504, то есть возврат во второе меню. В этом меню какие-либо действия возможны только при подтверждении Вашего желания - нажатии "Y".

Немного изменена логика работы блока "ON ERROR GO TO" и в подпрограмме GO SUB 20 - "нажмите любую клавишу". В строке 22 RANDOMIZE 22 заменено на RANDOMIZE 24, то есть при нажатии на любую клавишу, в том числе и "BREAK", работа программы будет продолжена.

Нулевая строка и коды, размещенные в ней, в основном такие же, как и в дебюте программы "PROG". Отличие лишь в том, что вместо SOUND1 - SOUND2 находятся SOUND1 - SOUND20. Это демонстрационные примеры тех звуков, которые могут быть созданы при помощи этой программы.

Для получения блока кодов "sound1-20" можно воспользоваться программой:

```
10 CLEAR 43775: PRINT "PLEASE WAIT..."
20 LET a=43776: LET s = 0
30 FOR n=1 TO 20
40 RESTORE 200
50 FOR a=a TO a+31: READ y: POKE a, y: LET s=s+y: NEXT a
60 NEXT n
70 LET a=43776: RESTORE 300
80 FOR p=1 TO 8: READ m: LET s=s+m
90 FOR n= 1 TO 20: READ y: POKE (a+32*(n-1) +m),y: LET s=s + y: NEXT n
100 NEXT p: CLS
110 IF s<>57522 THEN PRINT AT 0, 0; FLASH 1; "ERROR" : STOP
120 SAVE "sound1-20"CODE 43776,640
130 STOP
200 DATA 14,0,6,0,33,0,0,197,17,0,0,229,205,181,3,225,17,0,0,237,0,193,16,239,62,0,12,
     185,32,228,0,201
300 DATA 3,5,100,40,50,40,30,25,30,5,50,15,20,20,40,5,7,8,20,4,5
310 DATA 5,244,200,5,1,144,232,32,136,232,20,136,16,200,32,232,160,16,136,32,40
320 DATA 6,1,0,0,0,1,3,3,19,3,0,19,39,0,3,3,15,39,19,78,0
330 DATA 9,70,10,5,1,3,3,5,5,50,15,3,5,9,1,100,4,1,5,8,3
340 DATA 17,40,2,10,100,10,20,7,5,50,50,15, 200,144,200,50,100,244,244, 5,5,40
350 DATA 18,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0
360 DATA 20,82,90,9090,90,90,90,90,82,90,82,82,90,90,82,82,82,82,82,90
370 DATA 25,1,1,5,2,3,1,2,1,2,2,3,1,2,5,2,5,5,1,1,80
1000 FOR s=1 TO 20: RANDOMIZE USR (43776+32*(s-1)): PAUSE 20: NEXT s
```

Если все набрано правильно, то после того, как Вы сделаете RUN, программа выдаст для записи на магнитофон блок кодов "sound1-20" длиной 640 байт. Вы сразу же можете прослушать Получившиеся звуки, сделав RUN 1000.

Вернемся к нулевой строке. Всего за оператором REM должно быть:

Симв. набор	768 байт
UDG-графика	168 байт
ON ERROR GO TO	73 байт
SOUND 1-20	640 байт
Всего:	1649 байт

Если вы уже набрали дебют программы "PROG", то можно поступить следующим образом. Сначала сохраним на ленте необходимые коды из нулевой строки "PROG":

```
SAVE "code" CODE 23760,1009
```

Это будут коды нулевой строки, кроме SOUND1-2. Дальше надо сформировать нулевую строку заданной длины. Когда Вы будете создавать такую строку при помощи программы "REM FILL", (см. "ZX-РЕВЮ", стр.197) то на запрос о числе добавочных байтов после REM Вы должны ответить: 1649. После того, как нулевая строка заданной длины будет сформирована, загружаем в нее коды:

```
LOAD "code" CODE 23760
LOAD "sound1-20" CODE 24769
```

Теперь эту нулевую строку можно соединить с текстом программы "SOUND" при помощи MERGE.

Рассматривая в этой и предыдущих статьях различные приемы, при помощи которых можно усовершенствовать простые программы на Бейсике, мы невольно начали касаться программирования в машинных кодах. Пока что это небольшие кодовые блоки, вызываемые из Бейсика. Но таких кодовых блоков может быть достаточно много. Вы и в дальнейшем будете выискивать в других программах (в частности в "SUPERCODE") интересные эффекты, получаемые при помощи машинных кодов и все больше использовать их в своих разработках. В результате постепенно будет создаваться Ваша личная библиотека. Важно

то, что для Вас уже не существует непреодолимой границы между Бейсиком и машинными кодами. Ваша программа станет все больше состоять из машинных кодов, которые в конечном итоге заменят всю Бейсик-программу, останется только Бейсик-загрузчик. Такой путь возможен, если идея, заложенная в программе, оказалась удачной и вышла за рамки "повседневных задач", с которых все началось.

Рассматривая некоторые приемы, я поделился лишь одним из огромного множества направлений, таким, которое в итоге приведет вас к полному переходу на програмирование в машинных кодах. А может быть, Вы выберете другой путь, например применение "диалектов" Бейсика - MEGABASIC, BETABASIC и т.д. Или компилирование Бейсик-программ. Каждый свободен в своем выборе.

* * *

Уважаемые читатели!

"ИНФОРКОМ" уже не первый раз печатает работы, присланные нашим постоянным корреспондентом Алексеевым А.Г. Мы надеемся, что многие из предложенных автором идей помогли Вам или еще помогут взять максимум того, что может дать Ваш "Спектрум".

Творческий поиск не стоит на месте и мы, конечно же, не в состоянии опубликовать все возможные идеи и новые разработки, а стараемся останавливаемся на вопросах, имеющих, как нам кажется, массовый интерес и широкое применение.

Для получения более специфичной информации мы можем предложить Вам напрямую связаться с нашим корреспондентом по адресу:

141220, Московская обл., Пушкинский район, п/о Черкизово-2, ул. Б.Тарасовская, д.113, кв.21. Алексееву Андрею Георгиевичу

При обращении просьба вкладывать заполненный конверт с обратным адресом.

В ответном письме Вы получите список авторских разработок служебных программ и вспомогательных драйверов, которые могут быть приобретены по почте за весьма скромную цену (плюс почтовые расходы).

Среди них, например, программный драйвер для программы AST-STUDIO, который позволяет использовать этот графический редактор вместе с принтером для тех, кто не имеет стандартных интерфейсов, поддерживаемых этой программой, а пользуется компьютерами с программируемым портом КР580ВВ55А или с интерфейсом LPRINT II/III. Таких пользователей у нас очень и очень много и до сих пор использовать графический редактор с принтером им было проблематично.

Этот драйвер позволит взять от редактора максимум: печать разномасштабных копий, печать как поперек листа, так и вдоль, подключать принтеры, имеющие и 8 и 7 иголок, получать полутоновые картинки с имитацией цвета оттенками серого.

Драйвер имеет небольшую длину. Вы всегда сможете набрать его самостоятельно и подключить к программе ARTSTUDIO, следуя подробным инструкциям.

Если у вас при этом нет оригинальной (настроечной) версии программы "ARTSTUDIO", а есть уже инсталлированная и неперенастраиваемая программа, этот вопрос тоже может быть решен.

Среди прочих разработок, предлагаемых Вашему вниманию, стоит отметить универсальный программный драйвер, позволяющий реализовать все графические возможности Вашего принтера. Будучи подключенным к текстовому редактору, такой драйвер позволит Вам формировать и распечатывать документы достаточно сложной фактуры. Это могут быть, например, фирменные бланки, виньетки, визитные карточки и т.п.

Исчерпывающую информацию и условия оплаты Вы получите, обратившись письменно по указанному выше адресу.

ВНИМАНИЕ

Уважаемые читатели!

"ИНФОРКОМ" проводит исследование, посвященное игровым программам для IBM-совместимых компьютеров.

Мы знаем, что многие из вас имеют доступ к IBM-совместимой технике в учебных заведениях, на работе, в клубах, через друзей, а некоторые и дома.

Нам очень ценно ваше мнение о значении игровых программ и мы будем очень рады, если Вы сможете написать нам несколько слов о самых любимых играх. Если Вы напишете названия нескольких любимых игр, по-возможности названия фирм, их выпустивших, и год выпуска - этого будет вполне достаточно, чтобы составить представление о том, как распространены эти программы в вашей стране и какой популярностью они пользуются. Свои отклики Вы можете отправлять на простой почтовой открытке.

Мы будем особенно признательны, если Вы сможете указать свой возраст и род занятий, поскольку изучение приверженности разных возрастных групп к разным игровым программам - одна из задач этого исследования.

Ваше мнение поможет в проведении этого интересного и, надо сказать, уникального эксперимента.

Ждем ваших откликов по адресу 121019, Москва, Г-19, а/я 16.

Если же Вас лично интересуют те исследования, которые мы ведем по игровым программам для IBM-совместимых компьютеров (а любители игровых программ оценивают их как очень интересные), то Вы можете ежемесячно знакомиться с ними на страницах журнала "МОНИТОР", который предоставляет для этого центральные полосы каждого номера. Планируется и выпуск отдельной книги массовым тиражом.

КАК ЭТО ДЕЛАЕТСЯ!

RANARAMA

По своему жанру эта игра относится к аркадным адвентюрам. Пожалуй, ярко выраженный мотив блуждания по огромному многоэтажному лабиринту все-таки приближает ее к чисто аркадным играм, но возможность использования различных объектов, встроенные аркадные вставки и применение магических заклинаний все-таки делают ее аркадной адвентюрой.

Автор игры - знаменитый программист из фирмы "HEWSON CONSULTANTS" - Стив Тернер. Наши читатели знакомы с его работами, например по программе "Quazatron", освещенной в "ZX-РЕВЮ-92" на стр.79, а также по серии публикаций "Профессиональный подход" в "ZX-РЕВЮ-91".

Мы знаем, что игры лабиринтного типа явились одними из первых игр на "Спектруме" и их период расцвета относится к 1982 -1983 годам. В какой-то мере можно удивляться, что в 1987 году на рынок поступило одновременно несколько продуктов, выполненных в единой манере лабиринтной игры - "Dandy" (ELECTRIC DREAMS), "Gauntlet" (US GOLD), "Ranarama" (HEWSON CONSULTANTS) и некоторые другие. Сразу несколько фирм вернулись к старой лабиринтной идее на новом уровне. Что это - случайность или закономерность?

Дело в том, что бурное развитие техники программирования к этому периоду позволило совсем по другому представить старую идею. В лабиринтных играх нового поколения мы видим трехмерную растровую графику, применение оттеночных эффектов для придания изображению глубины, многокрасочную цветовую палитру, совершенные звуковые эффекты и элементы стратегического планирования, необходимые для успешного прохождения игры и, конечно, значительное увеличение размеров игрового поля. Тому, как удается "втиснуть" все это ограниченный объем памяти "Спектрума" нам сегодня расскажет Стив Тернер на примере программы "Ranarama", а пока несколько слов о самой игре.

*   *   *

Начинающий чародей Мервин доэкспериментировался с различными магическими снадобьями до того, что превратил себя в лягушку. Положение, можно сказать, почти безвыходное, но нет худа без добра. Благодаря этому ему удается скрываться от своры враждебно настроенных колдунов, жаждущих его погибели (нам неизвестно за какие грехи).

Теперь он может обыскать восемь этажей в подземелье, на каждом из которых находятся по двенадцать колдунов, победить их и, вооружившись захваченными магическими приемами, вернуть себе человеческий облик.

Как только он находит колдуна и входит с ним в контакт, игра переходит в режим аркадной вставки. В этом режиме надо решить головоломку типа анаграммы в ограниченное время. Злой колдун перепутал буквы в названии игры "R.A.N.A.R.A.M.A" и, оперативно манипулируя джойстиком или клавишами, Вы должны расставить их по своим местам (этот элемент напоминает схватку двух роботов за обладание системой управления в предыдущей программе С. Тернера "Quazatron"). Работа требует сообразительности, глазомера и четкой координации движений. На каждом уровне время, отведенное для решения головоломки, постепенно уменьшается. Кажется, что ввод аркадных вставок в аркадно-адвентюрную игру стал как бы визитной карточкой фирмы "HEWSON CONS." (см. например игру "Firelord").

Если Вы с этой задачей справляетесь успешно, то программа возвращает вас в главный экран, колдун исчезает, и на его месте появляются магические руны, которые Вы должны подобрать как можно быстрее. Сбор этих рун является необходимым элементом, поскольку Вы сможете впоследствии конвертировать их в магические способности.

В пол лабиринта местами встроены иероглифы (сокращенно "глифы"), которые обладают разнообразными функциями. Так, например, "голова" обозначает "глиф колдовства" (Glyph of Sorcery"). Задействовав этот глиф, Вы можете проскроллировать и исполнить те магические заклинания, к которым получили доступ в результате сбора магических рун.

Имейте в виду, что некоторые заклинания не могут быть использованы, поскольку магические силы, которые с их помощью вызываются, могут быть доступны только на более высоких уровнях.

Поскольку в игре Вам дано только две "жизни", а этого явно маловато, то очень полезно приобрести заклинания, способные перебросить Вас на более высокий уровень.

Каждый из восьми уровней имеет от 50 до 100 комнат. В начале игры вас помещают в случайно выбранной комнате. Проходя через дверь, Вы как бы "включаете свет" в очередной комнате и она становится видимой. Очень полезно воспользоваться глифом "зрения", который позволит вам увидеть все комнаты своего этажа. Правда, вам покажут только те комнаты, в которых вы уже были, но зато есть возможность увидеть невидимые двери, которые при простом обходе комнат вы не обнаружили. Впрочем, их можно оригинально обнаружить и при обычном проходе по лабиринту. Движущиеся существа могут на мгновенье "просунуть" голову сквозь стену в тех местах, где есть такая дверь.

Как и должно быть в играх подобного рода, подземелье кишит всякой нечистью. Здесь Вы найдете змей, злобных гномов, чудовищных насекомых. Уничтожение их требует стрельбы и уменьшает запасы Вашей боевой мощи, но существует специальный глиф, с помощью которого можно уничтожить всех монстров в пределах комнаты. Этот глиф, в отличие от остальных, расходуемый и не восстанавливается после применения.

Значительно большую опасность представляют такие неприятные духи, как вращающиеся мечи или, например, щелкающие челюсти. Жизнь усложняется тем, что они не поражаются Вашим огнем. Надо подумать, каким образом уничтожить тот генератор, который насылает их на Вас.

Даже когда Вы пройдете все комнаты на этаже, Вы найдете, наверное не всех колдунов (а должно их быть двенадцать). Уничтожить всех необходимо, поэтому придется разыскать тех, кого Вы пропустили. Так как они шляются, где пожелают, то может быть Вы просто разминулись с ними по пути. Есть специальное заклинание, которое поможет обнаружить их местоположение - можете воспользоваться им.

Вот, пожалуй и все. Собирайтесь в путь и настройтесь на увлекательную игру, требующую как логического мышления и наблюдательности, так и определенных спортивных навыков. А мы сейчас предоставим слово Стиву Тернеру для рассказа о том, как создавалась эта игра.

* * *

В прошлых статьях из серии "Профессиональный подход" я немного рассказал о том, что делается на той кухне, где программисты готовят новые программы для Вашего "Спектрума". В основном это были теоретические статьи, а вот сейчас на примере программы "RANARAMA" мы посмотрим, как это происходит на практике.

Техническое задание.

Поскольку я свободный программист, то работаю над тем, что мне нравится, и вроде бы нет никакой необходимости в подготовке технического задания на разработку игры. Тем не менее, это не так. Просто-напросто я сам ставлю себе техническое задание и сам же себе его утверждаю. Это дисциплинирует и позволяет на всех этапах работы сверять то, что получается с тем, что должно было быть.

Концепция игры пришла от программы "Paradroid", написанной для "Коммодора-64". В принципе, эта концепция была использована и при подготовке программы "Quazatron", но способ экранного представления был там совершенно иным. Во-первых, сам сценарий там

был научно-фантастическим, а мне казалось, что игра на тему магии и волшебства очень хорошо впишется в структуру программы типа "Квазатрона". К моменту, когда я в принципе обдумал сценарий, стоящий за игрой, у меня уже довольно четко вырисовались требования к программе.

1. Достижение трехмерного образа на экране за счет применения теневых эффектов. К тому времени, когда я задумал эту игру, такая графика уже была испытана на "Коммодоре", но на "Спектруме" в полной мере этот прием еще не был освоен.

2. Отказ от скроллинга экрана и за счет этого возможность использования всего цветового многообразия компьютера, т.к. отпадает существенная часть атрибутных проблем (проблема "клэшинга" атрибутов).

3. Двумерное представление игрового поля на экране (вид сверху). Отказ от принципа "один экран - одна комната". Одновременно на экране могут изображаться несколько комнат. В то время это было новым словом.

4. Структура игры - аналогична программе "Quazatron", но в основе сценария должна лежать магия и волшебство. Соответственно, меняется тема и содержание аркадной вставки.

5. Принцип разделения "монстров" на две категории.

Первостепенные - колдуны. Найти и победить их - необходимый элемент для успешного исполнения программы.

Второстепенные "монстры" не являются необходимыми - это просто объект для стрельбы. Их не надо искать, они нападают сами. Количество их разновидностей измеряется десятками. Общее количество уничтоженных "монстров" за время игры - порядка сотен.

6. Техника постепенного высвечивания игрового поля. Впервые эта идея появилась во время тестирования программы "Quazatron", но туда она не пошла. Суть состоит в том, что карта игрового поля хранится "нераспакованной" для тех участков, которые еще не были исследованы, а на экране эти области затемнены. Это была пионерная идея. В то время аналогов еще не было.

7. Обработка "скрытого" изображения. Эта техника требует пояснения. Дело в том, что "монстры", находящиеся в комнатах, соседних с той, в которой находится "герой", на экране показываться не должны, но, тем не менее, должны жить собственной жизнью, перемещаться по заданным законам, в общем, вести себя так, как если бы они были видны.

Эта идея была заимствована у программы "Paradroid", но значительно переработана с целью ускорения обработки данных.

Косвенный эффект от применения такой техники - ускорение перестроения изображения на экране, т.к. все перемещения выполняются только в пределах ограниченного окна экрана, соответствующего комнате, в которой находится герой.

8. Стопроцентное использование площади экрана. К тому времени, о котором идет речь, очень широко распространилась техника выделения в качестве динамического "окна" какой-то части экрана, например одной трети или двух третей. При этом остальная площадь экрана заполняется статичной, неизменяющейся графикой или закрашивается чёрным цветом или используется для вывода текстовых сообщений.

На мой взгляд, использование всей площади экрана должно было дать особую конкурентоспособность программе, по сравнению с прочими продуктами, имеющимися на рынке.

9. Вышеизложенное техническое требование означало для меня полный отказ от текстовых сообщений и всяких статичных локальных дисплеев. Это, в свою очередь, вызвало требование так организовать пользовательский интерфейс, чтобы всю информацию о ходе игры пользователь получал бы через динамическую графику и звук.

10. Требования к логике управления "монстрами" заключались в том, что они должны вести себя "разумно". Они должны искать героя, переходить из комнаты в комнату и выходить за пределы экрана. Их способы перемещения по игровому полю должны быть достаточно мотивированы. То есть, они должны входить и выходить из помещений через двери, а не материализовываться из воздуха, когда и где вздумается.

Когда я подготовил такой набор технических требований к программе, я почувствовал реальность выполнения этой задачи как с точки зрения объема занимаемой оперативной памяти, так и с точки зрения быстродействия процессора.

Предварительные исследования.

1. Дизайн экрана.

Исследовательскую часть я начал с самого важного, на мой взгляд, момента, ибо он во многом будет определять коммерческую ценность будущей программы. Я построил демонстрационный экран программы и оценил:

- художественное впечатление;
- объём расходуемой памяти:
- быстродействие.

Всеми результатами я остался доволен. Особенно привлекательными мне показались добротные, прочные стены, разделяющие комнаты.

Все графические элементы, присутствуюшие на экране, конструировались из специальных символов, для которых я создал знакогенератор.

2. Раскладка оперативной памяти.

Закончив с первым этапом, я перешел ко второму. Поскольку у меня уже был определенный опыт, я поначалу воспользовался той картой оперативной памяти, которая сложилась после завершения программы "Quazatron", благо структура у этих двух программ была похожа. Я выделил участки памяти для хранения карты игрового поля, для хранения нужной мне графики, для всевозможных таблиц, массивов, для области рабочих процедур и программных переменных.

Как обычно, вскоре я обнаружил, что мои амбиции заходят слишком далеко и не все, что я запланировал, можно "втиснуть" в спектрумовское ОЗУ. Так, например, я хотел иметь 24 типа различных движущихся "монстров", но пришлось урезать их количество до 14.

3. Упаковка данных.

Убедившись, что оперативной памяти мне не хватает, я начал думать об упаковке данных. В первую очередь, мне предстояло хранить несколько карт игрового поля (по одной для каждого уровня) - это наиболее емкие данные.

Я подготовил несколько различных методов, поэкспериментировал с ними и, наконец, остановился на наиболее оптимальном.

В принятом мною методе на задание комнаты на карте игрового поля расходовалось всего два байта и еще по два байта уходило на описание каждой двери. Вот как это было сделано.

Единицей измерения на карте я выбрал блок из четырех знакомест (размер 16x16 пикселов).

Для комнат в первом байте я решил хранить координату левого верхнего угла комнаты, я во втором байте - размер этой комнаты. Вы знаете, что поскольку байт не может принимать значение больше 255, то мне нелегко было бы сделать достаточно большое игровое поле. Максимум - 16x16 блоков по 16x16 пикселов, то есть, чуть больше одного экрана. Это, конечно, недостаточно, поэтому координата левого верхнего угла задается не как абсолютная, а как относительная, то есть это "смещение" начала N-ой комнаты относительно N-1 -ой комнаты. Тогда все стало на свое место (пример см. на рис. 1)

Пришлось "помудрить" и со вторым байтом, задающим размер комнаты. В итоге я остановился на том, что у меня в программе будет ограниченное количество разных типоразмеров комнат и второй байт будет задавать собственно не размер комнаты, а номер ее типоразмера. Соответствующая рабочая процедура потом по этому номеру найдет в таблице данных истинный размер комнаты (пример см. на рис. 1).

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71

. . . . и так далее . . . .

ОРГАНИЗАЦИЯ ДАННЫХ ПО КОМНАТАМ

N комнаты	смещение	типоразмер
1	0	1
2	6	2
3	24	2
4	18	3

. . . .
и так далее

ТИПОРАЗМЕРЫ КОМНАТ

1 2 3

. . . . и т.д. . . . .

Рис.1

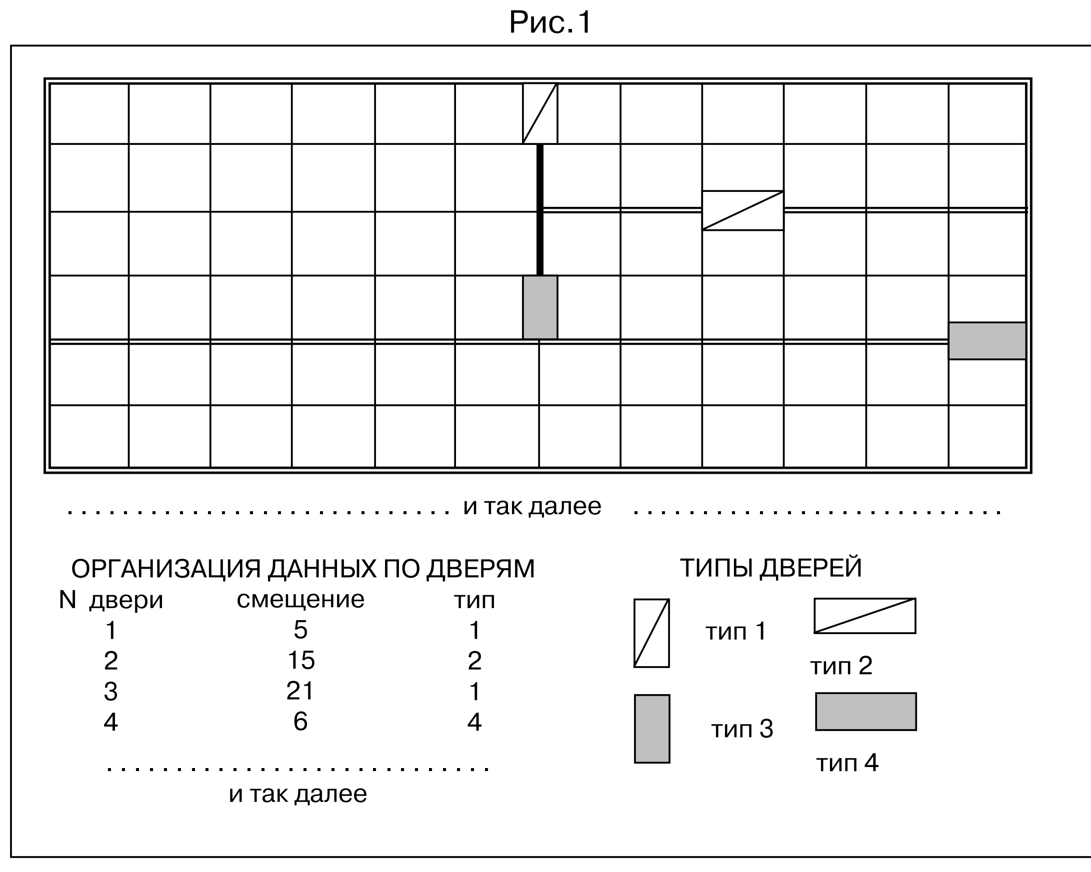


Рис.2

Аналогично было и с дверьми. Первый байт задает "смещение" координаты двери относительно координаты предыдущей двери. Второй байт задает тип двери. Двери могут быть четырех типов. Во-первых, они могут быть вертикальными (тип 1) или горизонтальными (тип 2), а во-вторых, они еще могут быть и невидимыми (типы 3 и 4 соответственно). Пример см. на рис. 2.

4. Специальные алгоритмы.

Для того, чтобы работать с упакованными, как было описано выше данными, пришлось разработать несколько специальных алгоритмов и на их основе создать несколько соответствующих процедур.

В основу была положена следующая логика работы (представим себе, что герой входит в новую комнату и ее надо "высветить" на экране):

а) По координате "героя" из упакованной карты игрового поля извлекаются координаты комнаты и дверей.

б) На экране строится соответствующая комната из символов, которые выдает специальный знакогенератор.

в) По номеру комнаты из специальной таблицы сценария извлекаются данные о наличии в ней объектов и предметов.

г) По их номерам и координатам они изображаются на экране.

д) Если в таблице программных переменных есть "монстры", находящиеся в данный момент в этой комнате, то изображаются и они.

Для создания трехмерного эффекта теневой графики я разработал специальный алгоритм, который назвал "шэдоу-процессором". После того, как комната и весь ее внутренний антураж построены, на изображение накладываются полутоновые горизонтальные и вертикальные тени. На рис. 3 доказано, как все тени изображаются с помощью всего лишь двух элементов.

"Шэдоу-процессор" позволил сэкономить еще изрядное количество памяти.

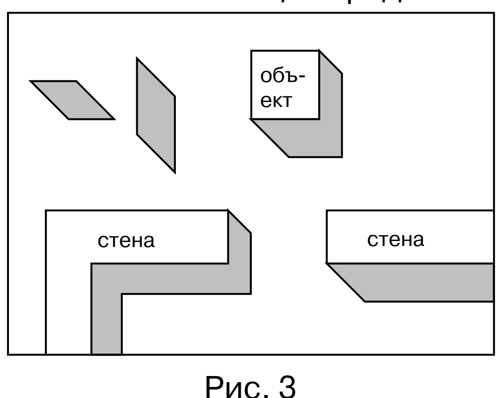


Рис. 3

5. Проверка концепции.

Предприняв такие меры, я успокоился относительно распределения оперативной памяти. Теперь я понял, что ее мне хватит, оставался открытым вопрос о достаточности быстродействия компьютера для того, чтобы обслуживать экран при работе по описанной выше логике.

Пришлось воплотить идеи в машинный код и проверить, как обстоит дело с быстродействием. Не сразу, но дело пошло. Работа оказалась неожиданно сложной. Только примерно через месяц я убедился в том, что экранное представление моих идей совпадает с тем, что я хотел получить. Теперь, закончив предварительные исследования и убедившись, что задача в принципе выполнима, я мог приступать к основному объему работ - разработке программы в целом. Надо сказать, что в этот момент мне пришлось преодолеть массу искушений добавить в программу что-то еще. Это было, конечно, возможно, но при таком подходе можно никогда не выйти из стадии предварительных исследований.

Дизайн программы.

Закончив с предварительными исследованиями, я перешел к проработке структурной диаграммы. В статьях "Профессиональный подход" (см. "ZX-РЕВЮ"-91) мы уделили серьезное внимание тому, как создаются структурные диаграммы, зачем они нужны и чем они отличаются от алгоритмических блок-схем. Там же мы упомянули и о том, что при всем многообразии игровых программ, структура их может оказываться удивительно похожей. Так случилось и в этот раз. Практически эта важная и ответственная работа превратилась в пустую формальность. Почти на 100 % подошла структура, ранее разработанная для

программы "QUAZATRON". Конечно мне помогло то, что я начал разработку новой программы не на пустом месте, а уже имея солидный опыт программиста игровых программ. Если же Вам когда-то придется начинать это дело "с нуля", то для справки я привожу структурную диаграмму на рис. 4.

Машинный код.

Этот этап я начал с того, что "перетащил" все процедуры, которые можно, из программы "QUAZATRON" в новую программу. Нет никакого смысла изобретать велосипед и если Вы абсолютно уверены, что та или иная процедура надежно работает и хорошо отлажена, то Вы не только сэкономите время на ее разработку, но и более того, она явится необходимой базой для создания работающих с ней совместно других процедур и для их отладки.

Сейчас у меня уходит на создание такой программы примерно восемь месяцев. Это много, но я абсолютно все делаю сам. Тем более для меня очень важна экономия времени за счет использования большого количества проверенных процедур. И, надо сказать, многое мне удалось использовать. Процедуры верхнего уровня структурной диаграммы были вообще перенесены с минимальными доработками.

Основную трудоемкость, как ни странно, составило не программирование машинного кода, а борьба с разного рода мелкими неприятностями. Примерно месяц ушел на то, чтобы выловить все "жучки" в ассемблирующей программе. Я пользовался ассемблером "OCP", а он регулярно разрушал мои таблицы меток. Еще столько же времени ушло на борьбу с механическими помехами (дребезг контактов на разъеме, через который подключались дисковод и принтер).

Но так или иначе, рано или поздно, наступает такой момент, когда перед Вами еще гора работы, а сделано уже так иного, что отступать нельзя. Вот на этой стадии мне и пришлось столкнуться с концептуальной проблемой, связанной с моим героем. Как оказалось, у меня на него осталось так мало места в оперативной памяти, что сделать приличного чародея я уже не мог. Как я ни экспериментировал, он меня не устраивал. Спас положение Эндрю Брейбрук, который предложил сделать героя лягушкой и тогда спрайт 16х16 получается намного лучше. В поисках приемлемой картинки для своей лягушки я перелистал несколько томов по биологии, неплохо ознакомился с жизнью земноводных и, самое главное, узнал, что лягушка по латыни звучит, как RANA. Так и родилось название игры RANARAMA.

Вторая проблема возникла, когда я уже перевалил за середину своего проекта. Я сделал аркадную вставку, в которой мой герой обменивался ракетными ударами со злыми колдунами. Выпущенная ракета могла взаимодействовать с другими, стационарными ракетами и инициализировать их. При правильной игре можно было вызвать что-то вроде цепной реакции и тогда героям приходилось бы непросто.

Все, кто видел мою работу на этой стадии, однозначно оценили эту аркадную вставку, как неудовлетворительную. У меня было еще очень много работы с графикой и я отложил решение этой проблемы в долгий ящик.

Однажды рано утром я сидел и экспериментировал с клочками бумаги на столе, пытаясь придумать оригинальное изображение титульного экрана с названием игры, выполненным крупными буквами. В этот момент меня осенило. Я понял, что аркадной вставкой может стать перестановка букв в слове RANARAMA. Как оказалось, название программы очень соответствует этой задаче. Поскольку в слове есть 4 буквы "A", то аркадная вставка получилась скорее динамичной, чем головоломной. Первые же эксперименты убедили меня в том, что это решение удачно. Труды предыдущего месяца были сняты с полки и полетели в мусорную корзину, а через пару дней проблема перестала существовать.

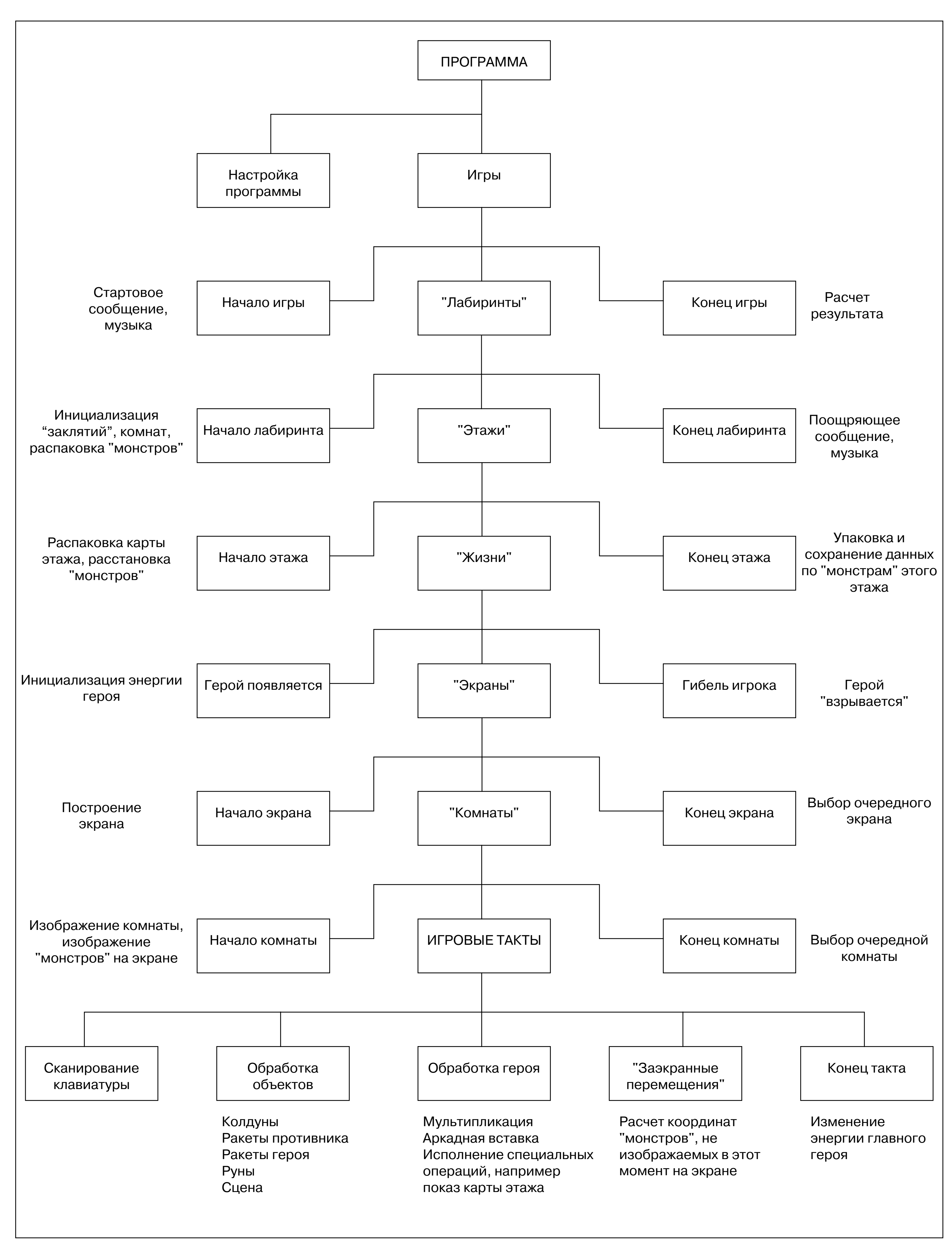


Рис.4 Структурная диаграмма программы RANARAMA

Звук и музыка.

Музыку в программу я добавляю в самый последний момент. Это происходит потому, что музыкальные и звуковые процедуры работают в режиме прерываний 2-го рода - IM2.

Примечание ИНФОРКОМа: В последнем издании книги, посвященной машинным кодам, мы широко рассмотрели применение прерываний 2-го рода. Это издание объединяет выпущенные ранее в 1990 году три тома и примерно на 20% дополнено. Общий объем - 271 стр. Заказы принимаются на приобретение отдельных экземпляров и на лицензированный тираж в регионе принимаются.

Поскольку меня очень часто спрашивают, как мне удается получать такие интересные звуковые эффекты в программах, я остановлюсь на этом вопросе поподробнее, хотя здесь не буду касаться работы с прерываниями 2-го рода, дабы не нарушить простоту изложения.

"Спектрум" имеет очень ограниченные звуковые возможности. У него есть всего лишь один звуковой канал, который может находиться в двух состояних - "вкл"/"выкл". Пульсирующий сигнал определенной частоты вызывает появление звукового тона соответствующей частоты (см. рис. 5).

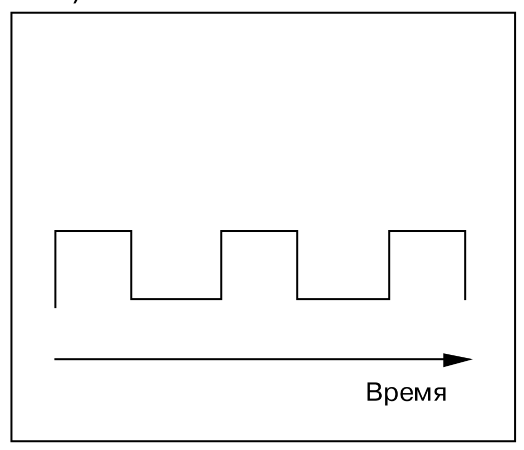


Рис. 5

Таким образом, создание сложных звуков сводится к управлению частотой.

Тот принцип, который мы здесь рассмотрим, может быть использован и владельцами 128-килобайтных машин при программировании встроенного звукового процессора. Я применял этот прием и для "Коммодора 64" при озвучивании программы "URIDIUM". Аналогичную технику используют и синтезаторы серии "Ямаха DX".

Предлагаемая Вашему вниманию программа выполняет линейную модуляцию частоты. Таким образом, частота изменяется вверх или вниз в соответствии с данными, взятыми из специальной таблицы. На рис. 6 и 7 показаны графики частот.

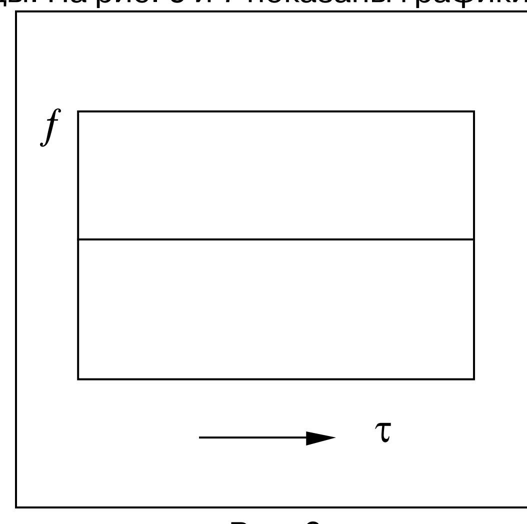


Рис. 6

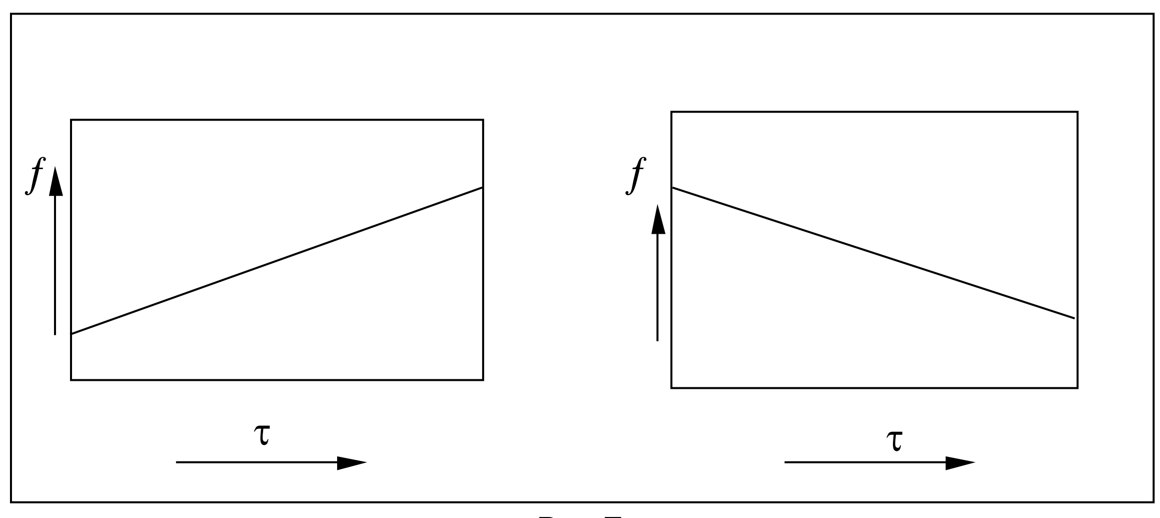


Рис.7

Одноступенчатая модуляция.

Итак, изменяя частоту звука вверх или вниз или и так и этак, мы можем сделать звучание более интересным. В приведенной ниже программе используется только изменение частоты по линейному закону. Это означает, что скорость изменения частоты остается постоянной. Вы можете поэкспериментировать и с другими формулами, меняя не только саму частоту, но и скорость ее изменения.

Пилообразная модуляция

(рис. 8) легко организуется путем создания в программе счетчика тактов. Всякий раз, как счетчик будет обнуляться, частота возвращается к своему исходному значению. Другой счетчик отсчитывает необходимое количество таких циклов. На рис. 8 видно, что пилообразная модуляция может быть как прямой, так и обратной.

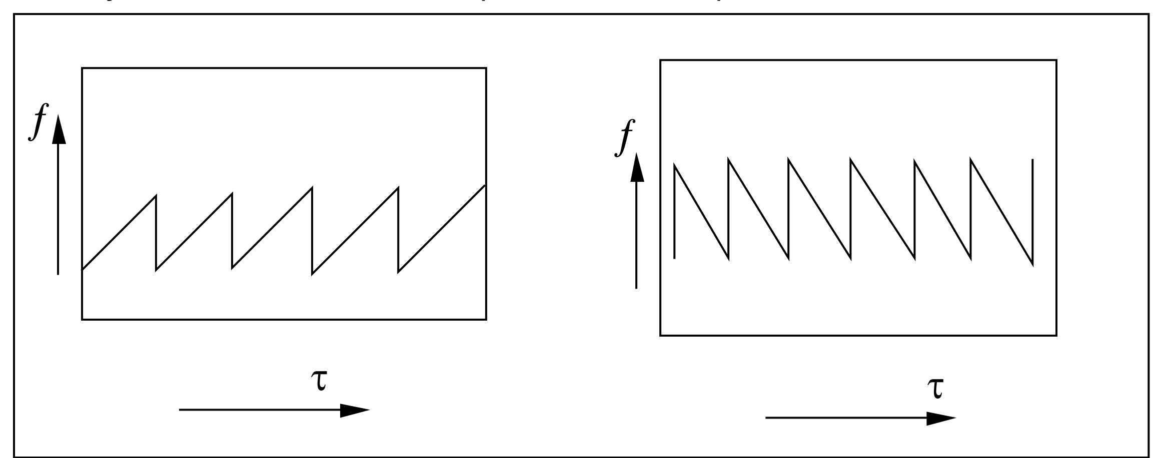


Рис. 8

Треугольная модуляция

(рис. 9) обеспечивается тоже с помощью счетчика тактов, но в этом случае при обнулении счетчика происходит не восстановление исходного значения частоты, а изменение знака ее приращения.

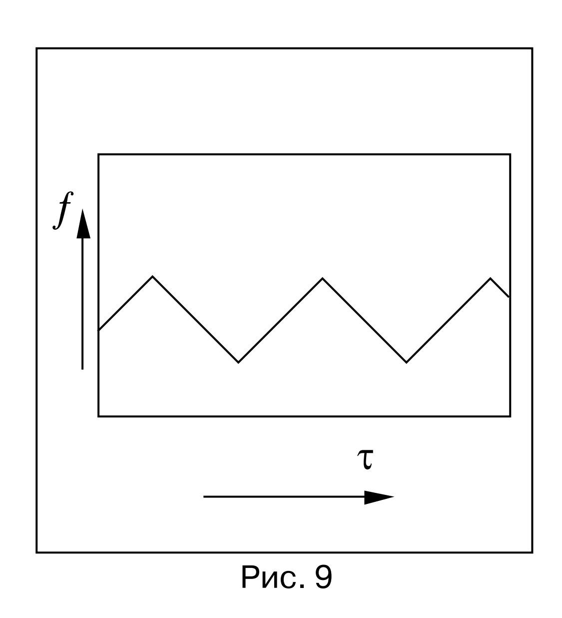


Рис. 9

Двуступенчатая модуляция

Пример такой модуляции показан на рис.10. Всякий раз, когда счетчик тактов обнуляется, возврат к исходной частоте происходит с некоторым смещением.

Все остальное - дело практики. Если идея Вам понятна, то можете приступать к экспериментам. Вы можете создавать собственные алгоритмы для управления частотой, можете хорошо поэкспериментировать с данными в таблице звуковых эффектов. Скоро Вы почувствуете диапазон Ваших возможностей.

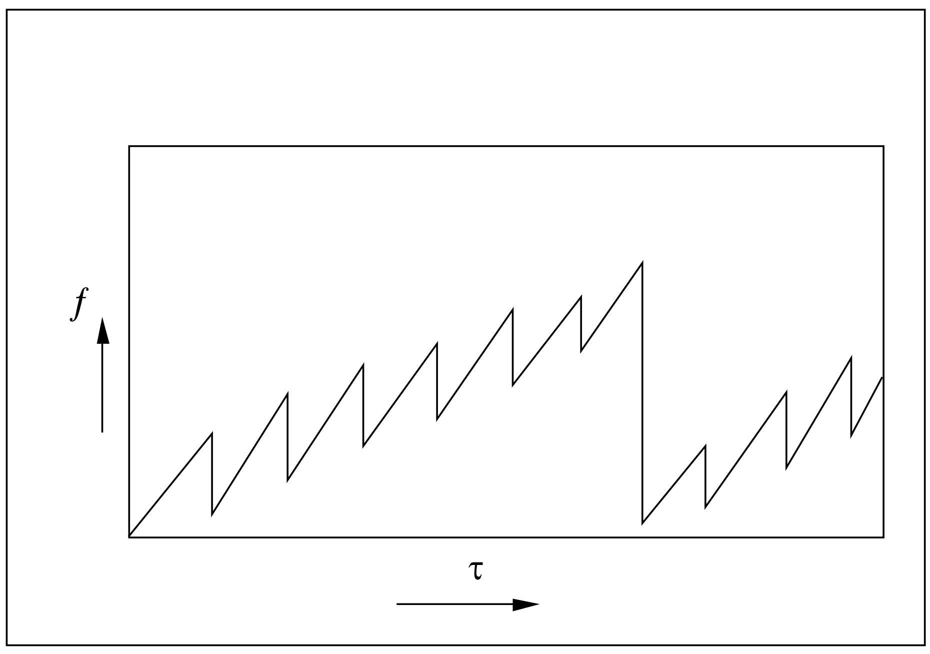


Рис. 10

Программа.

Вашему вниманию я предлагаю программу для испытания звуков. Назовем ее SOUNDTESTER. Она состоит из двух блоков. Первый блок - БЕЙСИК-загрузчик (Листинг 1). Он загружает второй блок и выполняет настройку параметров, которые Вы пожелаете изменить. Здесь Вы задаете закон, по которому будет происходить модуляция частоты, здесь же задаются настроечные параметры. Второй блок - в машинных кодах (Листинг 2). Он, собственно, и выполняет все необходимые расчеты и выдает сигнал на порт звукового динамика.

ЛИСТИНГ 1

```
10 CLEAR 40000
15 LET sontab = 45235: LET sonreq = 45234
20 LOAD "sound" CODE 45000
30 DIM a(10)
```

```
40 FOR x=1 TO 9: LET A(x)=0: NEXT x
1000 CLS: PRINT "          SOUNDTEST"
1010 PRINT ' "1. SOUND NUMBER", a(1)
1020 PRINT "2. START FREQ. ", a(2)
1030 PRINT "3. FREQ. CHANGE". a(3)
1040 PRINT "4. CHANGE TIMES". a(4)
1050 PRINT "5. REPEAT TIMES". a(5)
1060 PRINT "6. MODULATE TYPE",a(6)
1070 PRINT "    0 = SAWTOOTH"'
      "    1 = 2nd MOD DOWN"'
      "    2 = 2nd MOD UP"'
      "    OTHERS = TRIANGLE"
1080 PRINT "7. RESET FREQ", a(7)
1090 PRINT "8. CHAHGE T RESET",a(8)
1100 PRINT "9. CHAIN TO NO",a(9)
1110 PRINT "10. DELAY", a(10)
2000 INPUT "ENTER 0 TO FIRE SOUND OR NUMBER TO CHANGE VALUES"; i
2010 IF i = 0 THEN GO TO 3000
2020 IF i>10 OR i<1 THEN GO TO 2000
2030 LET x=INT (i)
2040 INPUT "NEW VALUE 0. . . 255 "; i
2050 IF i<0 OR i>255 THEN GO TO 2040
2060 LET a(x) = INT (i)
2063 IF x=1 THEN GO TO 2500
2065 REM Запись звуков в таблицу.
2070 LET z=sontab + (a(1)*8)+x-2
2080 POKE z,a(x)
2090 GO TO 1000
2500 REM вызов звука на редактирование
2510 LET z=sontab + (a(1)*8)
2520 FOR x=2 TO 9: LET a(x)=PEEK z: LET z=z+1: NEXT x
2530 GO TO 1000
3000 REM Вызов звука на прослушивание
3005 LET x=a(10)
3010 PRINT AT 20, 1; "PRESS 0 TO REPEAT SOUND OR 1 TO CHANGE. "
3020 POKE SONREQ, a(1)+1
3025 PAUSE 20
3030 RANDOMIZE USR 45000: FOR y=0 TO x: NEXT y: IF INKEY$="" THEN GO TO 3030
3060 IF INKEY$ ="0" THEN GO TO 3020
3070 GO TO 1000
9000 FOR x= 45227 TO 45400: PRINT PEEK x: NEXT x
```

КОММЕНТАРИИ

Строка	Содержание
1010	Номер звука а(1)
1020	Начальная частота а (2)
1030	Скорость изменения частоты а(3)
1040	Количество модуляций в звуке а (4)
1050	Количество повторений звука а(5)
1060	Вид модуляции а(6): 0 – пилообразная, 1 - двухступенчатая нисходящая, 2 - двухступенчатая восходящая, пр. – треугольная,
1070	Частота сброса а(7)
1080	Темп изменения частоты сброса а(8)
1090	К какому звуку "привязать" данный звук? а(9)
1100	Пауза а(10)
2000	Введите 0 для прослушивания звука или число от 1 до 10 для внесения изменений.
3010	Нажмите 0 для повтора или 1 для редактирования.

Листинг 2

```
                                ORG 0AFC8H
AFC8    F3          SOUND       DI
AFC9    3AB2B0                  LD A,(SONREQ)
AFCC    A7                      AND A
AFCD    281F                    JR Z,NONEW
AFCF    32B1B0                  LD (SONNOW),A
AFD2    FE0A                    CP 0AH
AFD4    2825                    JR Z,NOISE
AFD6    21B3B0                  LD HL,SONTAB        ; Адрес таблицы звуков
AFD9    3D                      DEC A
AFDA    87                      ADD A,A
AFDB    87                      ADD A,A
AFDC    87                      ADD A,A
AFDD    5F                      LD E,A
AFDE    AF                      XOR A
AFDF    32B2B0                  LD (SONREQ),A
AFE2    57                      LD D,A
AFE3    19                      ADD HL,DE
AFE4    010800                  LD BC,08
AFE7    11A9B0                  LD DE,SONFRQ
AFEA    EDB0                    LDIR
AFEC    1632                    JR PROCESS
AFEE    3AB1B0      NONEW       LD A,(SONNOW)
AFF1    A7                      AND A
AFF2    CA8EB0                  JP Z,SONEX
AFF5    FE0A                    CP 0AH
AFF7    E027                    JR NZ,PROCESS
AFF9    1609                    JP CNOIS
AFFB    3E0A        NOISE       LD A,0AH            ; Звук 10.
AFFD    32ACB0                  LD (SONLEN),A
B000    AF                      XOR A
B001    32B2B0                  LD (SONREQ),A
B004    0630        CNOIS       LD B,30H
B006    CD90B0      GAIN        CALL RANDOM
B009    E610                    AND 10H
B00B    D3FE                    OUT (0FEH),A
B00D    0E02                    LD C,02H
B00F    0D          MAKE        DEC C
B010    20FD                    JR NZ,MAKE
B012    10F2                    DJNZ GAIN
B014    21ACB0                  LD HL,SONLEN
B017    35                      DEC (HL)
B018    2074                    JR NZ,SONEX
B01A    AF                      XOR A
B01B    32B1B0                  LD (SONNOW),A
B01E    166E                    JR SONEX
B020    3AA9B0      PROCES      LD A, (SOFRQ)
B023    67                      LD H,A
B024    3E10                    LD A, 10H
B026    16FF                    LD D,0FFH
B028    5C          SONLP       LD E,H
B029    D3FE                    OUT (0FEH),A
B02B    EE10                    XOR 10H
B02D    15          FREQ        DEC D
B02E    2805                    JR Z,MOD
B030    1D                      DEC E
B031    20FA                    JR NZ,FREQ
B033    18F3                    JR SONLP
B035    3AAAB0      MOD         LD A,(SONCFG)
B038    64                      ADD H
B039    32A9BC                  LD (SONFRQ),A
B03C    21AEB0                  LD HL,SOKMOD
B03F    35                      DEC (HL)
B040    C28EB0                  JP NZ,SONEX
```

```
B043  21ACB0             LD HL, SONLEN
B046  35                 DEC (HL)
B047  2011               JR NZ,MODIFY
B049  AF                 XOR A
B04A  32B1B0             LD (SONNOW),A
B04D  3AB0B0             LD A,(SONNEX)
B050  A7                 AND A
B051  CA8EB0             JP Z,SONEX
B054  32B2B0             LD (SONREQ),A
B057  C38EB0             JF SONEX
B05A  3AAEB0   MODIFY    LD A,(SONRSF)
B05D  1F                 LD C,A
B05E  3AADB0             LD A,(SONTYP)
B061  A7                 AND A
B062  2820               JR Z,RESET         ; Треугольная
B064  3D                 DEC A
B065  2815               JR Z,TYP1
B067  3D                 DEC A
B068  E80A               JR Z,TYP2
B06A  3AAAB0   TYPOTH    LD A,(SONCFQ)      ; Пила.
B06D  ED44               NEG
B06F  32AAB0             LD (SONCFQ),A
B072  1614               JR MODE
B074  0C       TYP2      INC C              ; Двуст.восх.
B075  0C                 INC C
B076  79                 LD A,C
B077  32AEB0             LD (SONRSF),A
B07A  1606               JR RESET
B07C  0D       TYP1      DEC C              ; Двуст.нисх.
B07D  0D                 DEC C
B07E  79                 LD A,C
B07F  32AEB0             LD (SONRSF),A
B082  1600               JR RESET
B084  79       RESET     LD A,C
B085  32A9B0             LD (SONFRQ),A
B088  3AAFB0   MODE      LD A,(SONRND)
B08B  32ABB0             LD (SONMOD),A
B08E  FB       SONEX     EI
B08F  C9                 RET
B090  22A5B0   RANDOM    LD (RNHLST),HL
B093  2AA7B0             LD HL,(RNSEED)
B096  23                 INC HL
B097  7C                 LD A,H
B098  E603               AND 03
B09A  67                 LD H,A
B09B  32A7B0   ROK       LD (RNSEED),A
B09E  ED5F               LD A,R
B0A0  AE                 XOR (HL)
B0A1  2AA5B0             LD HL,(RNHLST)
B0A4  C9                 RET

B0A5  0000     RNHLST    DEFW 0000
B0A7  0010     RNSEED    DEFW 1000H
B0A9  00       SONFRQ    DEFB 00
B0AA  00       SONCFG    DEFB 00
B0AB  00       SONMOD    DEFB 00
B0AC  00       SONLEN    DEFB 00
B0AD  00       SONTYP    DEFB 00
B0AE  00       SOBRSF    DEFB 00
B0AF  00       SONRND    DEFB 00
B0B0  00       SONNEX    DEFB 00
B0B1  00       SONNOW    DEFB 00
B0B2  00       SONREQ    DEFB 00
B0B3  00       SONTAB    DEFS 256
B1B3                     END
```

СОВЕТЫ ЭКСПЕРТОВ

Сегодня в разделе, посвященном игровым программам, мы продолжаем печатать экспертные проработки нашего корреспондента из С.-Петербурга Фокина С.А., посвященные авиаимитаторам.

ATF

"Digital Integration" 1988

Эксперт Фокин С. А.
г. С.-Петербург

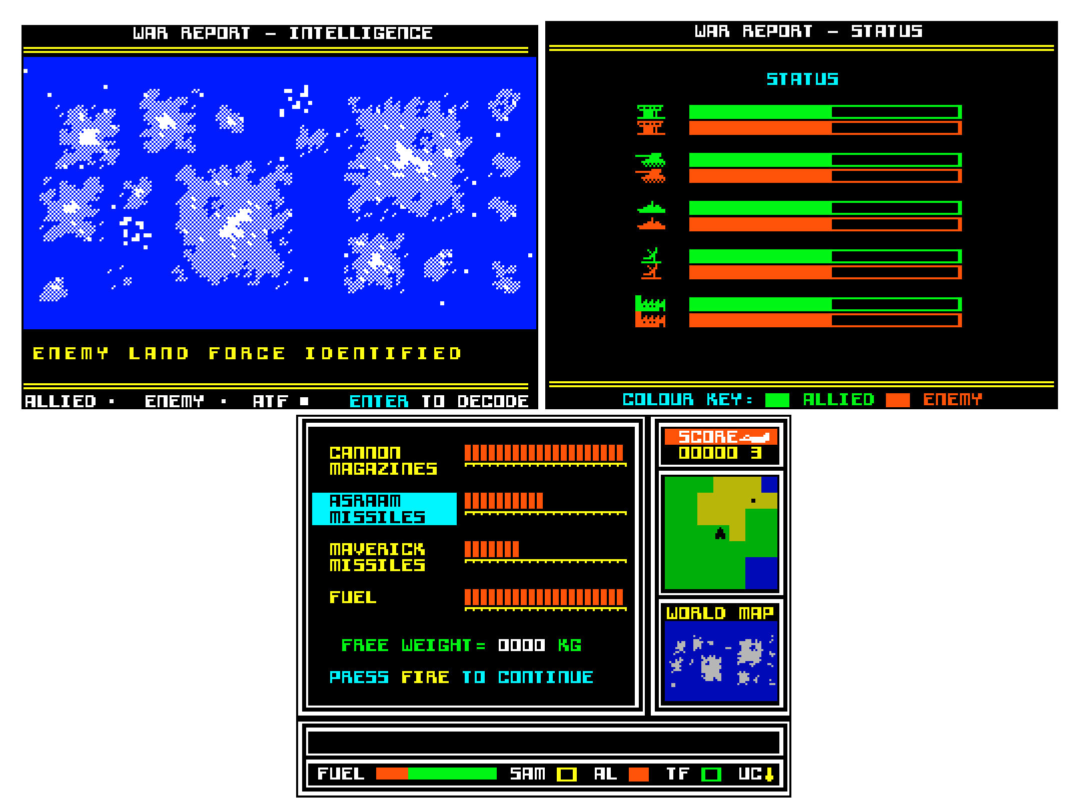


Глядя на название фирмы, сразу вспоминается такой серьезный имитатор воздушного сражения, как "TOMAHAWK". И "ATF" - это тоже имитатор воздушного боя, правда Вы будете управлять своим истребителем не из кабины, а как бы со стороны, но можете быть уверены, что игра от этого ничего не потеряла. Ваш тактический истребитель "ATF" будет летать над плавно перемещающейся поверхностью суши и океана, уничтожать вражеские перехватчики и производить пуски ракет по наземным и надводным целям, осуществлять посадки на автопилоте, пополнять боеприпасы и горючее и, в конечном счете, он должен выиграть сражение.

После загрузки программы на экране появится основное меню, в котором Вы можете выбрать тип управления, рейтинг пилота и звуковое сопровождение. Указатель перемещается курсорными клавишами (6 и 7), выбор пункта осуществляется клавишами "0"или "ПРОБЕЛ". После старта на экране появится карта боевых действии, на которой можно наблюдать расположение вашего истребителя, союзнических и вражеских объектов. Здесь же выводится сообщение о типе индентифицированных вражеских объектов, которое заносится в базу данных бортового компьютера самолета. Для продолжения - нажмите "ОГОНЬ". Затем на экране появятся результаты расчета соотношения сил борющихся сторон. Выход также осуществляется клавишей "ОГОНЬ". После этого Вы можете выбрать необходимое Вам оружие суммарной массой в пределах 6000 кг. Вам будут предоставлены магазины для скорострельной пушки, ракеты "воздух - воздух", управляемые ракеты "МЭЙВЕРИК" класса "воздух - земля" и, конечно, горючее. Выбор снаряжения производится клавишами "ВВЕРХ", "ВНИЗ", увеличение или уменьшение количества - клавишами "ВПРАВО" - "ВЛЕВО" в соответствии с выбранной системой управления игрой.

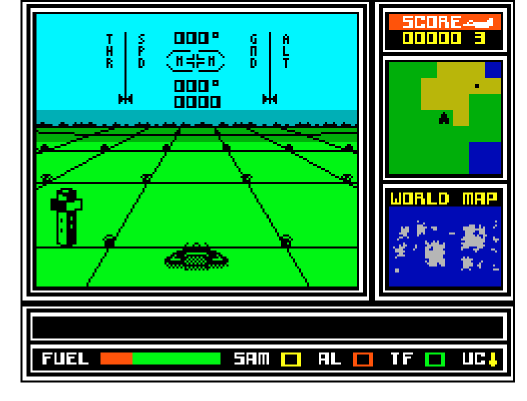


Оснащенный и заправленный самолет начинает выполнение полетного задания на посадочной площадке одной из Ваших баз.



На основной экране Вы видите 4 указателя: слева - обороты двигателя (THR) и датчик скорости (SPD), справа - расстояние до земли (GRD) и высота над уровнем моря (ALT). Между ними находятся индикатор прицела выбранной ракеты, над которым расположен указатель азимута курса истребителя, а под ним - указатели азимута и дистанции до выбранного объекта. Справа от основного экрана расположены два табло: на верхнем изображена окружающая обстановка, а нижнее табло мы рассмотрим более подробно. В исходном состоянии на нем изображена карта с указанием местоположения Вашего истребителя и выбранного объекта. Если Вы нажмете клавишу "C", то на табло появится выбранный бортовым компьютером объект. Переключение на объекты противника производится клавишей "D". Клавишей "E" можно пересмотреть все различные типы целей, а если их несколько, то все цели данного типа можно просмотреть клавишами "R" или "F". При дальнейшем нажатии на клавишу "C" на табло выводятся данные о запасах вооружений и боевой статус вашего истребителя. Темная полоса под экраном - это бегущая строка, на которую будут выдаваться текущие сообщения и предупреждения. А в самом низу слева находится индикатор горючего, справа от него расположен индикатор предупреждения о старте ракеты "земля - воздух" (SAM), который дублируется звуковым сигналом. Еще правее находится индикатор зоны действия базы (AL) (когда он мигает, можно предоставить исполнение посадки бортовому компьютеру).

Автоматизированная посадка задействуется клавишей "L". Следующий индикатор (TF) указывает на низковысотный режим полета, когда бортовая аппаратура ведет ваш истребитель и он плавно огибает рельеф поверхности земли. Этот режим включается клавишей "U".

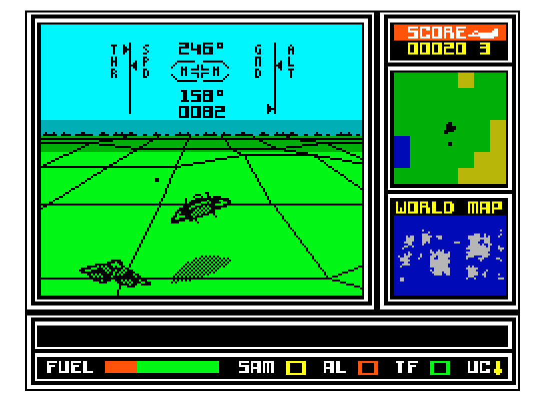


Ознакомившись с органами управления самолета и с бортовыми приборами, Вы можете смело произвести взлет. Клавиши "Q" и "A" отвечают за обороты двигателя. Набрав необходимую скорость, можно произвести отрыв от земли. Вам доставит истинное удовольствие процесс полета в игре, но не увлекайтесь, ведь воздушное пространство заполнено перехватчиками врага, а с каждым попаданием двигатели теряют мощность и скорость падает.

Итак, Вы выбрали наиболее близкую к Вам цель и на предельной скорости пошли на перехват. Время от времени индикатор "SAM" будет информировать Вас о наведении вражеской ракеты. Чтобы избежать катастрофы, вам нужно срочно, клавишей "J", запустить

генератор волновых помех. Когда до цели осталось меньше ста миль, Вы можете произвести пуск ракеты клавишей "M", но все же не рекомендуется стрелять ракетами с такого большого расстояния, т.к. эффективность попадания существенно снижается, и Вам придется пускать еще несколько ракет для полного уничтожения цели.

В момент пуска азимут цели и азимут курса должны совпадать, иначе попадание также будет не очень эффективно, если оно вообще будет.



Выбор типа ракеты выполняется клавишей "N", при этом на основном экране происходит смена формы прицела: с буквой "M" для ракеты " МЭЙВЕРИК", с буквой "A" для ракет "воздух - воздух". Для большей наглядности вы можете отключать сетку ландшафта клавишей "SYMBOL SHIFT".

Чтобы выиграть сражение, нужно как можно оперативнее уничтожать вражеские объекты, появляющиеся в базе данных компьютера. Если Вы уничтожили все объекты, не успокаивайтесь, т.к. через некоторое время будут идентифицированы новые, о чем будет выдано предупреждающее сообщение бегущей строкой. Не забывайте также, что горючее имеет тенденцию расходоваться, да и ракеты тоже. Поэтому время от времени Вам придется наводиться на ближайшую базу дружественных сил и, войдя в ее зону действия, предоставить посадку автопилоту. При посадке не следует забывать, что у истребителя есть выпускающееся шасси. После остановки самолета на экране появится информация об уничтоженных объектах, полученных в бою повреждениях истребителя, данные о его технических характеристиках и соотношении сил союзников и противника. Далее все продолжается по знакомой схеме: вы получаете информацию на карте боевых действия, проверяете соотношение сил борющихся сторон, пополняете боеприпасы и горючее и поднимаете свой самолет на задание. Вы достигнете победы, когда полностью уничтожите три каких-либо класса объектов. Если это Вам удастся сделать, Вы получите сообщение о победе.

В заключение хотелось бы отметить, что не надо полностью доверять автопилоту при полете на низкой высоте, т.к. иногда он не успевает отслеживать резкие изменения рельефа при максимальной скорости, и Вы можете задевать за вершины холмов. И еще: советуем не уклоняться от перехватчиков врага, а уничтожать их, т.к. чем больше Вы их уничтожите в начале игры, тем легче Вам будет в конце.

Желаем успеха!

Примечание ИНФОРКОМа: автор не указал, какой клавишей следует выпускать шасси самолета при посадке, а у нас, к сожалению, нет под рукой этой програмны, чтобы сделать проверку.

Начинающим пилотам мы рекомендуем проверить клавишу G (от слова GEAR). В большинстве испытанных нами имитаторов именно она выполняет эту функцию. Возможно, что после приземления следует затормозить самолет, обычно для этого служит клавиша B (BRAKES - ТОРМОЗА).

FLYER FOX

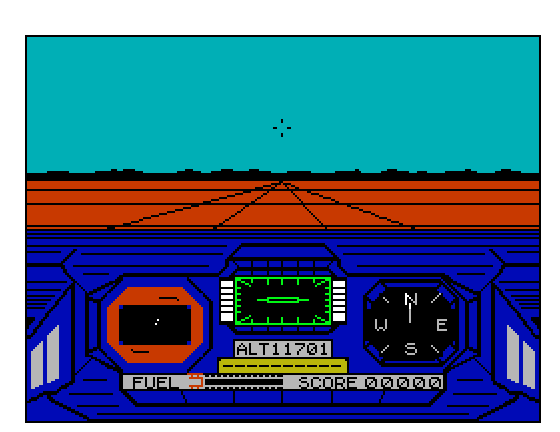
FLYER FOX

"Bug Byte", 1984

Эксперт Фокин С.А.
г. С.-Петербург

Эта игра хоть и относится к имитаторам воздушного сражения, но она настолько элементарна по своим задачам, что ее можно отнести к разряду обычных "стрелялок". Вам предстоит стать пилотом самолета сопровождения, которому поручено охранять авиалайнер от нападения истребителей врага. Все внимание вражеских летчиков приковано не к вам, а к авиалайнеру, но это не значит, что Вы можете сидеть сложа руки.

Когда программа загрузится, на экране появится заставка, в которой Вам необходимо выбрать свой вариант управления, а после этого нажать "ENTER".



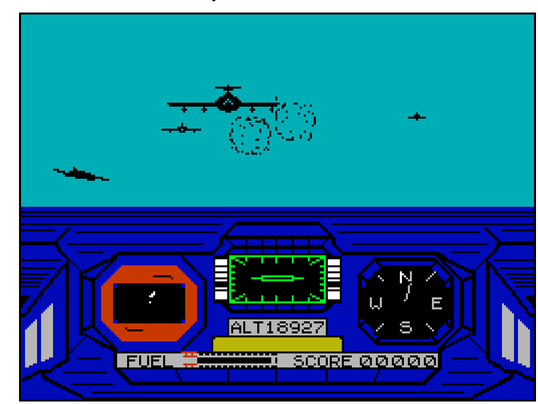
Игра запускается с 1-го уровня сложности. Вы взлетаете и пристраиваетесь позади авиалайнера. Приборная панель очень проста: в центре панели вверху находится авиагоризонт, под ним - альтиметр, а еще ниже - указатель повреждений авиалайнера. Справа расположен компас с указателем курса, слева - экран радара. В самом низу находится указатель уровня топлива и счетчик очков.



Программа несложно управляется:

"Q","Z" - вверх, вниз;
"I","P" - влево, вправо;
от "V" до "M" - огонь;
от "H" до "ENTER" - пауза (снятие паузы производится любой клавишей управления);
от "A" до "G" - включение/ выключение звука
"W" - окончание игры.

Через некоторое время после начала полета компьютер выдаст звуковое предупреждение о приближающейся атаке. Ваша задача - сбить истребители как можно раньше, пока они не успели причинить повреждения авиалайнеру. Повреждения подразделяются на четыре категории: незначительные, серьезные, глобальные и окончательные. На приборной доске постоянно указывается характер текущих повреждений. Кроме того, Вам выдается звуковое сообщение о полученных повреждениях.



На первом уровне сложности самолетов врага не очень много и, чтобы перейти на следующий уровень, Вам надо сбить 4 самолета. На втором уровне самолетов противника уже больше и надо сбить 5 самолетов, чтобы перейти на третий уровень и т.д. За каждый сбитый самолет Вам насчитывается 100 очков. За переход на очередной уровень присваиваются дополнительные очки. На этом, собственно говоря, сюжет игры себя исчерпывает, и нам остается только порекомендовать эту программу младшим школьникам, поскольку несложная система управления не отвлекает детей от содержания игры.

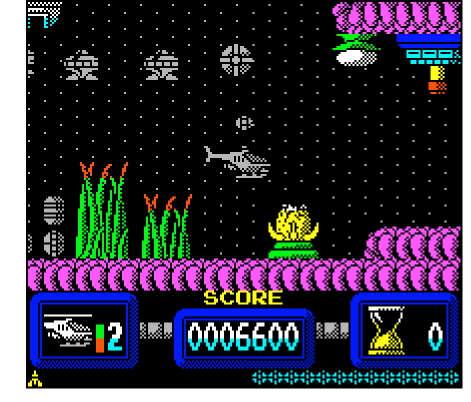
COBRA FORCE

"Players", 1989

Эксперт Фокин С.А.
г. С.-Петербург

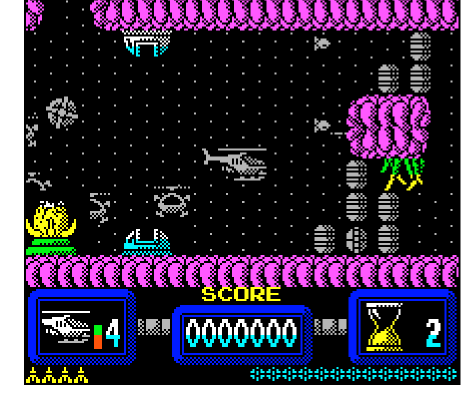


"COBRA FORCE" - типичный "боевик" с несколькими уровнями сложности. Пройдя удачно один уровень, Вы автоматически переноситесь в следующий. Так уж получилось, что самолеты и вертолеты - самые популярные боевые машины и вот в этой игре вам предстоит управлять вертолетом в чрезвычайно жесткой боевой обстановке. Мало того, что Вас встретит неимоверно плотный огонь противника, у вас будет очень мало свободного пространства. В общем, авторы игры немного перестарались и дойти до конца практически невозможно, если не... А вот что можно сделать, Вы узнаете в конце этого описания.



Итак, после загрузки на экране появится меню, в котором Вы можете сами назначить клавиши: "ОГОНЬ", "ПОДЪЕМ", "ВЛЕВО", "ВПРАВО" и "БОМБЫ". Затем, выбрав свой вариант управления, Вы переходите на первый уровень. Ваша задача (на всех уровнях) - расчистить плацдарм от наземных пусковых установок неприятеля, в то же время ваш вертолет подвергается непрерывным атакам с воздуха, со стороны летающих объектов, которые как быстро появляются, следуя друг за другом цепочкой, так же быстро и исчезают.

В левом нижнем углу экрана, рядом с изображением вертолета, указано количество "жизней" и текущий уровень энергии. В правом нижнем углу - количество бомб блокирующего действия (на некоторое время останавливаются все объекты на экране, кроме Вашего вертолета). В самом низу экрана расположены два индикатора: слева - количество ракет, которыми нужно уничтожать пусковые установки, Справа - количество оставшихся в этой зоне контейнеров, которые нужно собрать.



Пусковые установки можно уничтожить только ракетами. Чтобы их выпустить, необходимо нажать клавишу "ОГОНЬ" и задержать ее чуть дольше, чем это необходимо для простого огня из пулемета. Запасы ракет можно периодически пополнять. Для этого необходимо перехватить "звездочку", которая иногда попадается в строю нападающих на Вас объектов. После попадания, она превращается в ящик с боеприпасами, и Вам нужно успеть его подхватить.

Заканчивается расчистка плацдарма появлением двух вертолетов противника. Их можно уничтожить как ракетами, так и из пулемета. Только после этого зона считается расчищенной, и Вы переходите на очередной уровень. Уровни отличаются друг от друга только графикой, которая (надо отдать должное автору программы) становится все интереснее.

На этом можно закончить общее описание игры, но мы обещали дать одну небольшую хитрость, которая поможет Вам в этой тяжелой битве. Итак, если в самом начале, при переназначении клавиш, Вы наберете имя автора программы "SIMON", то получите неограниченную жизнь. Но, даже с такой поддержкой, не всякому хватит терпения довести игру до конца.

Желаем удачи!

THUNDER BLADE

U.S. GOLD, 1988

Эксперт Фокин С.А.
г. С.-Петербург



Многоуровневая (восемь уровней сложности) игра "THUNDER BLADE" -это типичный представитель жанра ACTION ("боевик") с прекрасной графикой и неожиданными поворотами сценария.

Небольшое, но весьма враждебное государство готовится нанести ядерные удары по нашей территории и спровоцировать третью мировую войну. Необходимо разрушить его стратегические объекты до того, как ракеты покинут пусковые шахты. Все попытки наших специальных подразделений по диверсионной работе потерпели неудачу. Командование приняло решение осуществить операцию по ликвидации вражеских объектов с помощью эскадрильи штурмовых вертолетов. Тем, кто смотрел такие фильмы, как "THUNDERBOLT-1" и "THUNDERBOLT-2", не надо объяснять, на что способен современный боевой вертолет огневой поддержки. Если же вы их не смотрели, то не отчаивайтесь, перед Вами игра, где Вы примете самое непосредственное участие в сценарии.

Вам предстоит занять место первого пилота (портрет справа). Можете взять себе напарника оператора управления оружием (портрет слева), тогда он будет нажимать клавишу "ОГОНЬ", но можете действовать и в одиночку. Ваш вертолет вооружен ракетами и шестиствольной пушкой, что вполне достаточно для успешного выполнения задания. Всего для выполнения задания у Вас имеется 6 вертолетов (5 силуэтов внизу и сам действующий вертолет на экране); совсем немного, учитывая плотность огня противника.



После загрузки нескольких первых блоков игры, ввод с магнитофона приостанавливается и выдается запрос на тип игры:

1 - CHEAT MODE.

2 - NORMAL MODE.

Это меню ввели не авторы программы, а Билл Гилберт, "сломавший" программу. В "хитром" варианте CHEAT MODE Вы имеете бесконечное число вертолетов.

Настоятельно рекомендуем пока им не пользоваться, а нажать клавишу "2" и продолжить нормальную загрузку. Для начала поиграйте в авторском варианте, чтобы

прочувствовать всю дьявольскую задумку создателей игры. Вам вряд ли удастся пройти хотя бы пару первых уровней. После этого, вероятно, Вы перезагрузите игру и отдадите должное хаккеру Биллу Гилберту, предусмотревшему упрощенный вариант. При бесконечном числе вертолетов у Вас появится шанс угробить пару сотен из них, но в конце концов все же успешно выполнить миссию.

Самое сложное в этой игре, как ни странно - освоить управление. В начале игры компьютер запросит тип управления. Если Вы выбираете клавиатуру ("1"), то управление следущее:

"O" - влево; "P" - вправо;

"Q" - вниз; "A" - вверх;

"SPACE" и "CAPS SHIFT" - огонь.

Чтобы войти в игру, достаточно нажать "ОГОНЬ". Вы окажетесь в городе, заполоненной боевой техникой неприятеля: самоходными зенитными установками, которые ведут по Вам прицельный огонь. Иногда между зданиями появляются патрульные вертолеты и они тоже стремятся вас сбить.

Вы можете набрать высоту (клавиша "ВВЕРХ") или опуститься вниз (клавиша "ВНИЗ"), а также перемещаться влево и вправо. Однако, чтобы набрать скорость, надо одновременно со снижением нажать клавишу "ОГОНЬ", естественно, перед этим набрав высоту. Чтобы снизить скорость или вовсе зависнуть на одном месте, необходимо нажать "ОГОНЬ" одновременно с набором высоты.

Если Вы выбираете в качестве органа управления джойстик, то компьютер выдаст сообщение:

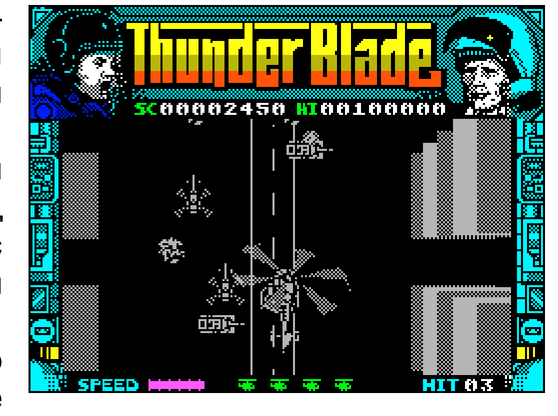
A - управление скоростью джойстиком;

B - управление скоростью от клавиатуры.

Если Вы нажимаете "A", то управление аналогично работе с клавиатурой. Во втором случае скорость вертолета можно изменять клавишами:

E - увеличение;

D - уменьшение.



В этой игре Вам предстоит уничтожить 4 стратегических объекта: атомный крейсер, пусковой комплекс, взлетающий космический корабль и командный пункт. Чтобы добраться до каждого из них, Вам предстоит преодолеть полосу обороны, которая воздвигнута перед этим объектом. Полоса перед крейсером - город, следующая - каньон, далее - каналы с боевыми катерами, и наконец, перед командным пунктом - опять город.

В начале описания мы не зря упомянули о неожиданных поворотах сценария игры. В каждой полосе обороны вид сверху неожиданно сменяется объемным изображением, хотя управление остается прежним. Преодолев полосу обороны и, оказавшись над стратегическим объектом, не спешите радоваться - он также хорошо прикрыт от налетов с воздуха. Многочисленные пусковые установки можно уничтожить, ведя непрерывный огонь, когда они показываются перед Вами. Здесь управление меняется: скорость набрать невозможно, она постоянна, а клавиши "ВВЕРХ" и "ВНИЗ" служат не для изненения высоты, а для маневра по вертикали. Впрочем, следует учесть, что вернуться назад можно недалеко.

Когда вы достигнете своей цели - командного пункта противника, то вертолет вообще не движется вперед, а может лишь уворачиваться от шквального заградительного огня. Не стоит пугаться - смело расстреливайте пусковые башни из всех видов оружия и командный пункт вскоре будет уничтожен.

Хочется надеяться, что вы долго будете вспоминать эту прекрасную игру.

В заключении приведем некоторые полезные советы.

1. Столкновение со зданиями в городе смертельно опасно, но некоторые из них можно пролететь, набрав максимальную высоту.

2. Не пытайтесь поразить все цели, это приведет к неоправданным потерям, помните, что ваша главная задача - преодолеть полосу обороны.

3. Т.к. огонь зенитных средств сосредотачивается на вашем курсе, то старайтесь лететь не по прямой, а меняя направление полета.

4. После потери очередного вертолета, скорость приходится набирать заново.

5. Появляющиеся над равнинной местностью эскадрильи самолетов почти не причиняют вреда, чего не скажешь о патрульных вертолетах.

6. При полете над стратегическими объектами также не старайтесь поразить все пусковые установки, помните, что Ваше главное оружие маневр.

Желаем удачи!

SANXION

"Chalamus" 1989

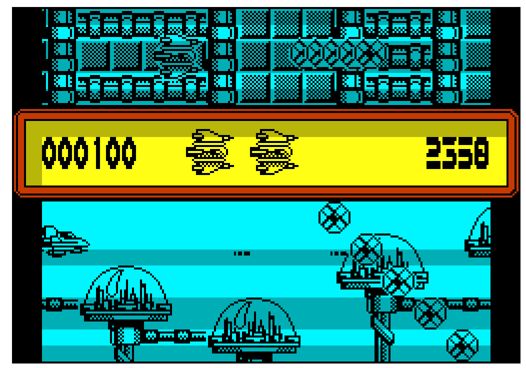


Эксперт Фокин С.А.
г. С.-Петербург

Игра "SANXION" - типичная "стрелялка" с неплохой графикой. Космический крейсер "Темное пламя", находящийся на станционарной орбите над неизвестной планетой, доставил на нее три хорошо вооруженных атмосферных разведчика типа "Москит", управляемых самыми опытными пилотами. Им предстоит совершить разведывательный полет над поверхностью планеты, населенной враждебно настроенной цивилизацией негуманоидного типа.

Технические устройства злобных аборигенов атакуют все, что появится в пределах их видимости. Беспилотный разведчик землян, перед тем как был сбит, успел передать на крейсер изображения странных поселений под прозрачными куполами и механизмов, летающих между ними. Дальнейшие попытки вступить в контакт с инопланетянами успеха не имели. Тогда капитан и отдал приказ послать на планету эскадрилью "Москитов". Одним из пилотов взяли Вас.

При запуске игра представляет широкий выбор типа управления. Если нажать клавишу "2" (клавиатура), то дается возможность самому назначить все клавиши управления - вверх, вниз, влево, вправо и огонь. Игра имеет очень простую систему управления. Для старта достаточно нажать "1".



Вверху экрана есть полезное приспособление - радар. Он дает вид сверху и показывает врагов вокруг вашего "Москита" еще тогда, когда их нет на экране. Советы играющему просты: в игре приходится надеяться не столько на свою огневую мощь, сколько на высокую маневренность. Ведь некоторые враги ведут по вам такой интенсивный огонь, от которого можно только уйти, а уничтожаются они не с одного выстрела. Следует стараться держаться середины экрана, т.к. нередко атаки следуют сзади или сверху.

От строя черных шаров, летящих зигзагом, можно просто уйти, поднырнув под них (они не ведут огня, а уничтожить их можно только при многократном попадании), некоторых врагов не обязательно полностью истреблять: покружив вокруг Вас и постреляв, они полетят дальше. Других приходится полностью уничтожать, иначе они от Вас не отстанут.

Следует помнить, что смертельно опасно не только попадание вражеского снаряда, но и само столкновение с врагом. Есть интересная особенность: если уничтожить в третьей волне врагов пару верхних или нижних и стать на их место, примерно на одной вертикали с оставшимися, то можно лететь так довольно долго. Вы будете недоступны для снарядов,

которые летят под углом 45 градусов. Однако надо не прозевать тот момент, когда они все-таки от Вас отстанут, метко выпустив в Вас пару снарядов на прощание.

Нам кажется, что эта игра доставит Вам много удовольствия. С каждой новой попыткой Вы будете прорываться все дальше и дальше, изучая повадки врагов. Правда, возникает мысль, что неплохо бы увеличить число самолетов раз эдак в пять.

AIR WOLF

"Elite" 1986

Эксперт Фокин С.А.
г. С.-Петербург



Игра "AIR WOLF" ("Воздушный Волк") по жанру относится к типу "ACTION", требует молниеносной реакции и фантастической точности.

В далеком Тибете, среди недоступных гор, была обнаружена база инопланетян, построенная ими при посещении Земли много тысячелетий тому назад. По всей базе то тут, то там спрятаны информационные карты, содержащие технические знания, неизвестные землянам. Каждая уважающая себя держава посылает экспедицию за экспедицией, чтобы завладеть этими бесценными сокровищами. Однако любому здравомыслящему человеку понятно, что ставший обладателем этих карт получит опасное преимущество перед остальным человечеством и сможет использовать его в злых целях. Узнав об этом, свихнувшийся миллиардер, помешанный на идее облагодетельствования всех на свете, решает нанять опытного пилота, который бы на маленьком, но хорошо вооруженном вертолете проник бы на базу и уничтожил бы все карты с опасными знаниями. Пусть уж лучше они не достанутся никому, чем поставят все человечество на край гибели. Вот этим пилотом Вы и станете.[1]

Сложность заключается в том, что хотя самих инопланетян на базе и нет, они оборудовали ее всевозможными ловушками и устройствами, открывающими огонь по любой приближающейся цели. Во время игры Вы встретитесь с силовыми полями и радарами, излучающими неизвестную смертоносную энергию, с электрическими разрядниками, с лазерными установками и еще со многим, что неизвестно землянам, но несет гибель Вашему вертолету.

Управление игрой простое:
клавиши от "Q" до "T" - вверх,
от "A" до "G" - вниз,
"X" - вправо,
"Z" влево,

---

[1] AIRWOLF™ Scenario: As Stringfellow Hawke, a former Vietnam chopper pilot, and the only man in the free world trained to fly the billion-dollar helicopter, 'AIRWOLF' you have been assigned a dangerous rescue mission by the FIRM. Five important U.S. scientists are being held hostage deep in a subterranean base beneath the scorching Arizona desert. As Hawke, you must guide AIRWOLF using full stealth capabilities, on a series of perilous night-time missions and bring about the release of each scientist in turn. Only destruction of the defense control boxes strategically positioned within the cavern will allow AIRWOLF to descend to the heart of the base where the scientists are held. (Прим. OCR)

"C" - огонь.

Миссия начинается с маленького аэродрома в горах. Посадочная площадка на этой картинке - единственное место, где Вы можете беспрепятственно посадить вертолет. Во всех остальных местах, при попытке сесть он разбивается о скалы. Поэтому надо постоянно поддерживать его на лету. В пещерах он вообще не может касаться никаких стен и предметов.

На втором экране вас поджидает силовое поле, появляющееся прямо перед вертолетом. В нем можно пробить проход, непрерывно стреляя из пушки. Но следует спешить: через некоторое время генератор полностью его восстанавливает.



На следующем экране такое же поле перекрывает путь вашему вертолету в пещеру, где и находится база. Самый простой способ преодолеть его - это снизиться в предыдущей картинке почти до самой поверхности и лететь в экран с силовым полем. Оно появляется быстро, но не мгновенно (подвела инопланетян их техника). Учитывая это, необходимо, оказавшись над провалом, немедленно начать снижение, тогда, едва не коснувшись поля, Вы попадете в очередной экран.

Здесь установлены два пульсара, время от времени излучающие потоки энергии неизвестной природы. Летите вправо, и Вы наконец найдете первую из информационных карт (по виду она напоминает квадрат с крестом). Смело расстреляйте ее, и Вам добавят 75 очков, а в картинке с пульсарами появится проход в скале. Миновав его, Вы попадете в экран с двумя электрическими разрядниками. Чтобы преодолеть их, Вам понадобится незаурядная реакция и расчет. В следующем экране выстрелами проложите себе путь в полу пещеры и, избегнув снарядов автоматической пушки и электрическую дугу (опыт прохождения уже есть), попадете в следующий экран с инопланетной картой. Она защищена двумя излучателями антиматерии, но мы думаем, что Вы без труда уничтожите ее и вернетесь назад.



Теперь направо и, избегнув снарядов, проскочите в следующее помещение. Здесь вас уже поджидают старые знакомые: автоматическая пушка, электрическая дуга, а также - очередная карта.

Преодолейте заслон и Вы в следующем помещении. Но что это? Здесь злокозненные инопланетяне установили башню с чудовищным лазером, который уничтожает все вокруг себя! Но хочется надеяться, что если Вы добрались до этого помещения, то сумеете преодолеть эту преграду.

Больше нет смысла утомлять Вас описанием всех картинок, придуманных изощренным умом программиста, а лучше привести несколько полезных советов. Вертолет может стрелять не только прямо, но и под углом. Для стрельбы наискось вверх необходимо, чтобы вертолет находился в свободном падении (ни одна из клавиш не нажата), тогда он задирает нос, и можно вести огонь по диагонали. Кстати, только таким способом можно попасть в первую карту. Наискось вниз он стреляет, если лететь вперед и вниз - тогда нос опускается.

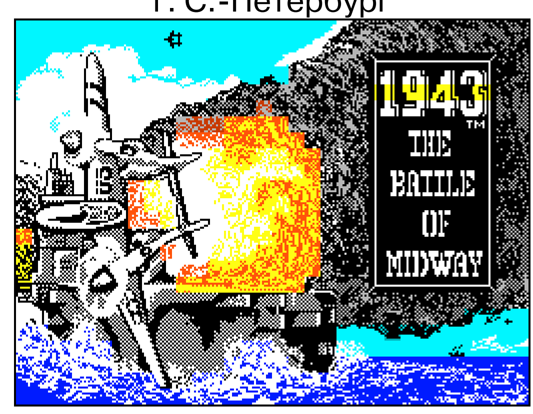
Вам не пройти далеко, если вы не научитесь стрелять вниз. Для этого необходимо повернуть вертолет в противоположную сторону, но вовремя отпустить клавишу, тогда вертолет станет "в фас". Кстати, такое положение полезно для проникновения в узкие щели. Хочется надеяться, что Вы приятно проведете время за этой игрой, если не повредитесь рассудком от неудач, ведь для прохождения всей игры у Вас всего 5 вертолетов.

Успеха Вам!

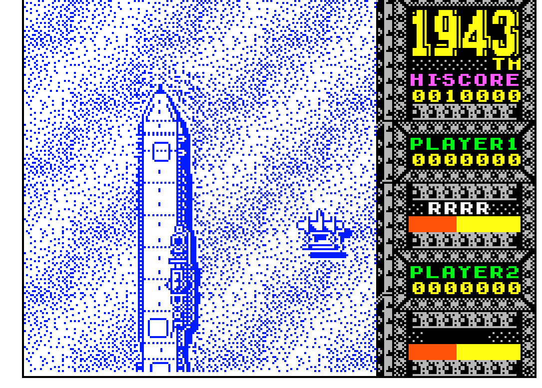
1943 THE BATTLE OF MIDWAY

"Capcom", 1986

Эксперт Фокин С.А.
г. С.-Петербург

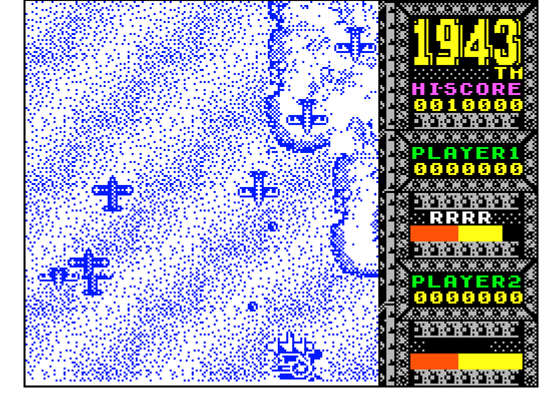


В этой игре, которую можно отнести к разряду обычных "стрелялок", Вам предстоит перенестись в годы Второй Мировой войны и вступить в бой с самолетами противника, а также с его военно-морскими силами. Вы будете управлять небольшим, но грозным истребителем, который подвергается нападениям вражеских истребителей, бомбардировщиков и обстреливается кораблями противника. Вся графика игры соответствует стилю рассматриваемой эпохи.



После загрузки программы нажмите любую клавишу, и внизу экрана появится бегущая строка. Вы можете выбрать свой тип джойстика или переназначить клавиатуру, а затем, в зависимости от того, сколько будет играющих, нажмите "1" или "2". Появится надпись, предупреждающая о готовности и Вы начнете свою миссию.

Игра разбита на несколько этапов, после прохождения которых она начинается сначала, но количество атакующих и интенсивность огня возрастают. Чтобы пройти этап и перейти к следующему, нужно пролететь определенный участок суши. Впрочем, это может быть и поверхность океана. Иногда на экране появляется большой четырехмоторный бомбардировщик, который будет стрелять в Вас до тех пор, пока Вы не повредите ему все четыре двигателя. Иногда, после удачного истребления нескольких самолетов, Вам будут попадаться контейнеры с надписью "POW". Если его перехватить, то Ваша энергия увеличится (индикатор энергии находится справа в средней части экрана), вы можете не брать этот контейнер, а расстрелять его. В этом случае вы получите новый, более мощный тип вооружения. Каждое попадание в контейнер "POW" приводит к переключению на другой тип оружия. Поражение вражеским снарядом вызывает утрату того, что вы приобрели таким путем.



Несмотря на то, что сюжет игры очень прост, в ней есть та изюминка, которая заставит Вас начинать все сначала и стрелять, стрелять, стрелять...

Желаем успеха!

*   *   *

Военными имитаторами и стратегическими играми интересуются многие наши читатели, но в то же время мы хорошо знаем, что среди них добрую половину составляют военнослужащие. Более того, "Синклер" в армии имеет популярность по-видимому большую, чем в других социальных группах. И для военных такие программы представляют не праздный, а профессиональный интерес. Сегодня у нас есть для них приятное сообщение. Редакционная коллегия журнала для военных профессионалов "Военный вестник" приняла решение о широкой поддержке пользователей компьютеров типа "ZX-Spectrum" в Вооруженных Силах. Первые публикации уже появились. Если Вам есть что сказать о возможностях использования Вашего компьютера в системе боевой подготовки, страницы журнала открыты для Вас. Ваши идеи, решения, разработки и программы будут опубликованы.

Это могут быть тренажеры, системы автоматического контроля уровня знаний, экспертные системы и многое-многое другое.

Кроме того, в редакционный портфель журнала входят интересные статьи, посвященные общетактическим и стратегическим проблемам. Любители компьютерных игр найдут здесь неисчерпаемый источник идей для создания собственных военно-стратегических программ.

Так, например, анализ операций типа "Буря в пустыне" - готовый план будущей игры. Разбор тактических целей наступающей и оборонявшейся сторон и анализ действий инженерных средств поддержки послужит для этого отличной базой.

Те, кто любят стратегические игры, знают, что большой интерес в них представляют вопросы тылового обеспечения. Ведь именно здесь военная стратегия сливается со стратегией деловой (STRATEGIC MANAGEMENT). Практически в каждом номере журнала вы найдете готовые алгоритмы для расчетов проблем, связанных со снабжением, пополнением, боепитанием, расходом сил и средств на прорыв обороны. Эти алгоритмы только ждут, чтобы их перевели на язык программирования и оформили в виде увлекательной игры.

Любители военной истории, военной техники, стрелкового оружия тоже найдут для себя много интересного. И, наконец, в последние годы лицо журнала представляют интересные и откровенные интервью с профессионалами. Такие интервью нечасто встретишь на страницах прочих газет и журналов даже в нашу эпоху всеобщей гласности.

Если у вас есть собственные идеи и разработки, редакция ждет ваших писем по адресу:

103175, Москва, Мясницкая ул., д.41, редакция журнала "Военный Вестник" Лушников Александр Васильевич, тел. (095) 896-79-49.

Желающие подписаться на "Военный Вестник", могут это сделать на почте. Индекс издания на 1993 год - 70140.

THE DARK WHEEL

ГЛАВА 3.

Если Вы хотите посетить кладбище на Тионисле, то лучше всего подлетать к нему со стороны солнца. Это и удобно и безопасно, поскольку политическая система Тионислы - Демократия и пираты здесь встречаются крайне редко. Тионисла выглядит из космоса ярко желтой планетой, а кладбище всегда расположено между планетой и звездой.

Первое, что Вы видите, когда подлетаете к нему - это серебряный диск и два спиральных рукава маленьких сверкающих точек. Это галактика в миниатюре. То же медленное вращение, тот же яркий блеск центральной части (именно здесь сосредоточены наиболее внушительные мемориалы).

Подлетев ближе, Вы увидите, что звезды в этой галактике вовсе и не звезды, а маркеры - внушительные металлические блоки, расписанные надписями на тысячах языков и символами тысяч религий. Это причудливое и волнующее зрелище. Маркеры редко бывают меньше, чем по тысяче футов в поперечнике. Среди них есть и хромированные кресты и титановые звезды Давида, а также всевозможные символы многочисленных миров и различных религий - порождения разума, которому довелось почить в этом необычном для космических путешественников месте.

У нижней границы этого необычного мавзолея расположена додекаэдрическая конструкция космической станции. Эта станция класса "Додо" - место жительства администрации и охраны некрополиса. Отсюда начинается путешествие по кладбищу, здесь служба безопасности проверяет Ваши документы и гостевую визу. Пока движется очередь прибывших на регистрацию, Вы можете любоваться замечательным зрелищем сквозь прозрачные стены и потолки станции. Здесь и там разбросаны разбитые и помятые корабли всех времен и народов. Ко многим пришвартована усыпальница, ставшая последним приютом для космического путешественника.

Существует масса причин для посещения кладбища на Тионисле. Многих притягивают те умопомрачительные сокровища, которые скрываются в недрах кораблей и саркофагов. Может быть и в этом черном кубе, порождении неземного разума, сделанном из непонятного металла скрываются вековые сокровища внеземных цивилизаций. Эх, если бы знать, как и на какую панель этого парящего саркофага нужно надавить, чтобы получить доступ к этим богатствам.

Но чаще включаются системы защиты. Это может быть скрытый лазер, а может быть робот-охранник с острыми, как бритва лезвиями вместо рук. Вас может всосать гиперпространственный вакуум и выбросить в другом измерении или в другой эпохе. И Вы, как и все прочие посетители, очень осторожно пробираетесь среди этих орбитальных руин, чтобы ничего не задеть и ни к чему не прикоснуться. Те, кто здесь похоронен, люди они или инопланетяне, так или иначе были очень и очень богаты. У них хватило средств приобрести здесь место последней стоянки для избранных и, конечно же, у них более, чем достало средств сделать свое убежище неприкосновенным.

Формальности завершены, новенькая, только что полученная пилотская лицензия проверена, туристский корабль довольно необычной формы выдан и Алекс готов к посещению кладбища.

Он поспешно отлетел от станции и, сверяясь с планом, начал разыскивать могилу Флейшера.

Он отыскал ее довольно быстро. Кем бы Флейшер ни был при жизни, он был чудовищно эгоцентричен. Его надгробием стала огромная кристаллическая структура, похожая на одуванчик. Каждая игла в этой конструкции сверкала, как чистый бриллиант и имела буквально сотни футов в длину. Его тело, облаченное в красную униформу бойца класса "Элита" парило в стасисе в самом центре этой конструкции, освещенное сфокусированными лучами солнца.

Рядом, у простого монумента, покоился пришвартованный корабль "Кобра". Все

жизненно необходимое оборудование: топливозаборники, боевые ракеты, грузовые отсеки, лазерные надстройки - было демонтировано.

Алекс внимательно пригляделся к кораблю. Он не имел ничего общего с той "Коброй", которая уничтожила отцовский корабль. Та имела все мыслимое вооружение, которое только можно купить за деньги.

Алексу показалось, что на корабле что-то мигнуло. Приглядевшись, он убедился, что это не обман зрения. Действительно, красный фонарик корабля сигналил кодом: "LAND ON DOR PL".

"Посадка на платформе" - легко расшифровал код Алекс. Сманеврировав легким челноком, он быстро подошел к "Кобре", пришвартовался на выгоревшей надстройке и виновато огляделся. Правила, действующие здесь, были очень суровы. Не то что швартоваться, но и прикасаться к любому сооружению на кладбище было смертельно опасно.

Пространство патрулировалось "Крейтами" службы безопасности, имевшими приказ расстреливать любого, независимо от пола и возраста, кто будет застигнут при попытке проникновения в гробницу.

К счастью, кладбище было слишком большим и тень многочисленных монументов создавала в этом городе мертвых достаточно безопасных мест для того, чтобы скрыться живым.

Входной люк открылся и зеленая лампочка призывно промигала: "Заходи". Алекс ввел свой туристический челнок в трюмное пространство и, когда загорелся сигнал "давление в норме", вышел из него и направился в рубку управления. Он открыл раздвижную дверь и на миг прищурился от яркого света приборов и панелей. Перед ним на широком экране сверкало изображение памятника Флейшеру.

Темным пятном на фоне яркого экрана вырисовывался силуэт мужчины, облаченного в космический костюм. Одна рука лежала на навигационной консоли, а другая - на кнопке боевого лазера.

- Я на борту, - сказал Алекс и подошел к молчаливому пилоту. Тот не сделал ни движения, не произнес ни слова.

На какое-то мгновение Алекс застыл, уставившись в экран, глядя на медленно перемещающиеся монументы и на сверкающие в черной пустоте звезды, а затем повернулся к хозяину корабля.

То, что он испытал, было похоже на шок и заставило отшатнуться. Перед ним было высохшее, мумифицированное лицо трупа. Оно смотрело на Алекса из под шлема и казалось, что высохшие губы растянулись в широкой улыбке.

- Как ты думаешь, стоит нам брать его с собой? - раздался голос откуда-то из-за спины.

Алекс удивленно обернулся и увидел фигуру, выходящую из тени.

- Как талисман на счастье.

Алекс попытался улыбнуться, но расслабиться ему не удалось. Слишком все это было быстро и неожиданно. Он как будто прирос к полу и смотрел на приближающуюся женскую фигуру.

Она была невелика ростом. Ее глаза были темными, а кожа имела оливковый оттенок. Одетая в светло-зеленые одеяния, она, кажется, утонула в них. Прикосновение ее руки было холодным и уверенным. На некоторое время она задержала свою руку в руке Алекса, глядя ему прямо в глаза и обезоруживающе улыбаясь.

- Итак, Рейф выбрал тебя? Хорошо, Алекс, во всяком случае путешествие с тобой будет по-крайней мере тихим. А у тебя вообще-то есть речевые функции? - с этими словами она шутливо развернула Алекса и похлопала по спине в поисках выключателя. - Или ты из старых моделей, которые умеют только жестикулировать?

- Простите, - сказал Алекс, - все это было так неожиданно для меня.

- О, боже, - сказала женщина - Где ты выключаешься? Мне кажется, молчащим ты был лучше.

- Кто ты? - спросил Алекс, слегка раздосадованный этим легкомыслием. Сейчас ему

больше всего хотелось бы знать, зачем Рейф Зеттер послал его сюда и куда он сам подевался?

- Я принадлежу к торговому союзу "Филдс"! - ответила она и в салюте приложила ребро правой ладони к левому плечу. - А зовут меня Элиссия Филдс. Имя несколько необычное, но ему я обязана своей приемной матери, которая в девятилетнем возрасте, когда инкубировала мой клон, увлеклась древнегреческой мифологией.

"Древнегреческая мифология. Инкубация клонов." - для Алекса это означало, что Элиссия происходит родом с планеты Теорг. Он напряг память, вспоминая все, что ему известно об этой планете.

Теорг. Обитаемая планета. Первое поселение основано двумя колонистскими кораблями. Экипажи кораблей приняли от местных жителей систему размножения посредством клонирования избранных особей. Все прочие - уничтожаются. На много столетий Теорг оказался отрезанным от общих путей развития цивилизации, от коммерции и торговли. По всей видимости, Элиссия Филдс - изгнанница.

- Меня зовут Алекс Райдер, - сказал юноша.

Я знаю, - ответила женщина и бросила на него взгляд, которым, казалось, можно было припечатать. Затем она похлопала по плечу тело, сидящее в кресле. - А это есть... или, вернее, был Генри Белл, космический торговец. Нам придется позаимствовать его гроб, хотя он, кажется, очень и очень этим недоволен. Это ржавое ведро было набито его трехмерными голограммами, которые рассказывали о том, как плохо будет тому, кто войдет сюда без разрешения. Большинство из них я повыключала, но возможно, что где-то что-то и осталось.

- Мы похитим этот корабль? - мягко спросил Алекс, изучая приборную панель. Топлива, судя по приборам, на борту было всего лишь на переход в 0,1 св. года. Этого явно недостаточно для того, чтобы покинуть систему Тионислы.

- Если ты предпочитаешь остаться здесь, то пожалуйста. Мы сможем ухаживать за могилкой, вырастим здесь цветочки и будем много-много разговаривать.

- Я имел в виду... как ты собираешься выбраться отсюда? - он уставился на инопланетянку. Кажется чувство постоянной душевной боли и горечи последних дней немного поутихло. Его явно интересовала эта девушка. - И почему, в конце концов, ты мне помогаешь? Где Рейф?

С отрывистым смехом Элиссия ответила.

- С Рейфом никогда ничего не поймешь. Ты можешь улететь на другой конец галактики, а он там уже тебя поджидает. Вернее, не он сам, а его трехмерная голограмма. А вот где он на самом деле, это нам еще предстоит выяснить. Что же касается помощи, то кто здесь сказал, что я тебе помогаю? Может быть, это ты помогаешь мне? А скорее мы помогаем друг другу. Ты собираешься отомстить за своего отца, мне тоже надо кое с кем, кое за что рассчитаться. Может быть, когда-нибудь я и расскажу тебе об этом. Во всяком случае, ты мне нужен, без тебя я не смогу вести этот корабль.

Алекс удивленно возразил - Но ведь "Кобра" управляется одним пилотом?"

- Да знаю я. И вообще я могу вести ее с закрытыми глазами. Не для этого ты мне нужен. Дело в том, что я с Теорга и потому мне здесь нечего делать. Мое лицо мгновенно вызовет массу подозрений. Ты нужен мне в качестве прикрытия, нам ведь придется много общаться с торговыми базами и официальными властями. Эта посудина совершенно безоружна и любой пират собьет ее с помощью палки для добычи бананов. Нам нужны защитные поля, ракеты, грузовые отсеки. Все это надо доставать, они ведь не растут на деревьях.

- Предлагаешь заняться торговлей? - спросил Алекс и мысль о длинной череде поколений, отдавших жизнь этому делу промелькнула перед ним.

Элиссия была права. Нечего и думать об охоте на "Кобру" без совершенного снаряжения, а сколько еще пройдет времени, пока будут закончены все юридические формальности с введением в права наследования, особенно если учесть, при каких странных обстоятельствах погиб отец.

Он чувствовал себя разрывающимся на части. Одна его часть хотела убить врага

прямо сейчас. Ему не терпелось вырваться на межзвездные трассы и броситься в погоню за убийцей. Другая часть подсказывала ему, что настоящий охотник должен иметь терпение. Поспешные шаги ничего не дадут, кроме провала дела. Нужен тщательно продуманный и спланированный подход, а хорошо вооруженный корабль - необходимый элемент всего предприятия.

- Все, что у меня есть - сотня кредов, - сказал Алекс, имея в виду ту сумму, которую ему выдали из страховки для возвращения домой.

- Это для начала. С этими деньгами мы начнем нашу торговую карьеру. - Ее лицо помрачнело, а в глазах засверкали отсверки огоньков приборной панели. - А затем мы полетим в такое место, о котором знает только Рейф Зеттер и устроим хорошую стрельбу. Мы разделаемся с кораблем, который убил твоего отца. Есть еще многое, за что он должен ответить....

Больше она ничего не сказала.

Для каждого, кто собрался заниматься космической торговлей, первой и самой трудной задачей является приобретение корабля. Конечно, вокруг каждой планетной системы есть свалки устаревших кораблей, проводятся и аукционы подержанных судов. Многие крупные компании нанимают вторых пилотов с гарантией расплаты через четыре года кораблем, если, конечно, новоиспеченный пилот останется к этому времени еще жив.

Но корабли очень и очень дороги, даже если приобретать их на свалке.

Алекса несколько смущала необходимость кражи этой посудины, но он не мог не оценить тех трудов, которые затратила смуглая беглянка, приводя в порядок корабль и запасая по каплям топливо и еду в количестве, достаточном для небольшого гиперпрыжка. В принципе все было готово, ей только недоставало партнера, который смог бы вести торговые операции, не вызывая подозрении в любом космопорту.

Они перетащили тело Генри Белла в туристический челнок и отправили его дрейфовать в пространстве.

- Так, отныне ты имеешь правовой статус "в розыске", - сказала Элиссия, как только они заняли места за приборной панелью. Но Рейф предполагает, что при уважительном отношении к телу этот статус не распространится за пределы Тионислы. Если бы мы уничтожили тело, будь уверен, нас начали бы искать во всех цивилизованных системах, а этого мы позволить себе не можем.

На экране видно было, как маленький челнок дрейфовал между огромными монументами. Алекс внимательно изучал показания сканеров и мониторов. Скромные запасы энергии позволяли включать только носовой и кормовой экраны. Да и для лазера этой энергии хватило бы только на один-два выстрела, а ракет, конечно, не было. Корабль по-прежнему находился в непосредственной близости от станции "Додо", положение которой хорошо просматривалось на трехмерной сетке навигационной карты.

Медленно "Кобра" развернулась и плавно двинулась вперед, в направлении границы гравитационного поля кладбища. Алекс внимательно следил за зеленоватым свечением сканера. Медленно, крадучись, проплывали на экране монументы и станционные суда.

- Я должна тебе рассказать кое-что об особенностях неуправляемого гиперперехода.

Алекса на мгновение передернуло.

- Спасибо, я уже об этом знаю. К тому же мы перелетим не более, чем на десятую светового года и, возможно, это не очень опасно.

Элиссия усмехнулась.

- В какого бога ты веришь?

- В Фактор Случайности.

- Я тоже.

Они посмотрели друг другу в глаза. Мимо проплывали монументы и монолиты. Звездное поле перед ними расширялось.

- Почти вышли, - вздохнула Элиссия. - Готовься к переходу.

Алекс бросил взгляд на сканер. Две ярких точки внезапно вспыхнули на экране и быстро устремились к ним.

- Эскорт!, - сказал он и Элиссия громко чертыхнулась.

- У нас мала энергия лазера, - сказал Алекс.

- Только попробуй его использовать и мы потеряем все шансы для торговли. Это же полиция! Корабли, может быть и не "Вайперы", но это все же полиция!

Пространство впереди уже почти очистилось. Корабли службы безопасности разделились и начали маневр охвата. Элиссия начала отсчет готовности к гиперпереходу

- Десять секунд.

"Кобра", отвыкшая за много лет покоя от активной работы, задрожала всем телом.

- Они приближаются, открыли огонь.

- Пять секунд.

"Кобра" заскрежетала, получив первый удар лазерного луча. Последние остатки энергии защитных полей исчезли. У атакующего корабля перегрелся лазер. Его партнер замешкался на мгновение, обходя монумент внушительных размеров, выстрелил неудачно и промахнулся.

- Три...

Они на огневой позиции... Залп приближается...

Оба корабля опять сошлись. Огонь их лазеров полыхал вокруг "Кобры".

- Два...

Еще удар, крик боли, корабль почти потерял управление и вдруг... ТУННЕЛЬ!

Элиссия откинулась в кресле. Алекс криво ухмыльнулся. Когда он глянул на свою спутницу, то увидел крупные капли пота на ее лице. Он протянул руку, пальцы мелко дрожали и ничего поделать с этим было невозможно.

ГЛАВА 4

- Так, теперь у тебя есть корабль и немного денег, - сказал Рейф Зеттер. - У тебя есть второй пилот и она прекрасный стрелок, лучший, чем ты сам, впрочем я надеюсь, что это не надолго. Теперь все зависит только от тебя, Алекс. И еще один совет. Если бы Джейсон был жив, он тебе сказал бы об этой сам. В минуту опасности отбрось разум, забудь о силе, забудь о правилах. Действуй так, как подсказывают тебе чувства. Если и это не поможет, то по крайней мере тебя не будет среди тех, кто будет сожалеть о том, как все закончилось.

Сидя перед навигационной панелью, Алекс смотрел на корабль Рейфа. Это была повидавшая виды "Анаконда". Грузовой отсек пробит, топливоприемники распахнуты, бортовые огни мигают и вовсе не потому, что это сигнал, а просто поскольку давно нуждаются в ремонте.

Рейф не пригласил Алекса на борт. В десятой доли светового года от Тионислы он оборудовал себе тихое убежище, где потихоньку потрошил разбитые корабли и собирал все, что может пригодиться для своего судна - механизмы, приборы, топливо и еду. Три небольших истребителя типа "Мамба" были пришвартованы к телу "Анаконды". Над ними возились роботы.

Вскоре после прибытия "Кобры" в частную систему Зеттера, его изображение появилось в рубке Алекса.

- Непростое дело оснастить и заправить корабль для такой миссии, которая тебе предстоит. Я заправлю вас достаточно для того, чтобы добраться до Айзинора, а дальше действуйте сами. Вам нужны ракеты, лазеры, энергетическая бомба, энергозаборники и многое другое. И не показывайся мне на глаза, пока не снимешь скальп с той гадины, которая убила Джейсона.

- Почему ты делаешь все это ради меня?

- Все это я делаю не ради тебя, а ради Джейсона, - ответил Рейф. - И ради многих других. И вот еще что, Алекс. Забудь о Ракксле. Не думай о ней пока. Время для этого еще придет...

- Но зачем же тогда отец назвал мне это имя?

- Чтобы ты передал его мне, чтобы я понял, что он в тебя верил, чтобы я знал, что он видел в тебе будущего бойца класса Элита. И его послание дошло.

Второй раз за сутки у Алекса закружилась голова. Да о чем говорит этот старик? Сначала отец оказался бойцом высшего класса, а теперь оказывается еще, что он видел

такой же потенциал в сыне. На имитаторах Алекс всегда имел высший балл и однажды даже получили большой приз - туристическую поездку в Фантастический город, копию легендарного Лондона. Но ему и в голову не могло прийти, что он когда-либо сможет подняться выше ранга "Опасный".

Но быть Элитой...

Головокружительные перспективы. Ему предстоит провести жизнь в битвах с пиратами, в полетах по самым опасным системам. Ему предстоит искать опасность, а не бежать от нее. Он должен широко оповещать о своем статусе самых опасных преступников и побеждать их в бою.

- Одно я могу сказать тебе точно, - продолжал Рейф. - Если ты не пройдешь через все испытания и не станешь Элитой, можешь и не думать о Раккслe, забудь о ней. И ты никогда не узнаешь, что искал твой отец.

- Не понимаю.

- Ты знал о его связях с лигой "Темного Колеса"?

Час от часу не легче. "Темное Колесо" было полулегендарным союзом звездопроходцев, которые объединились для того, чтобы исследовать, что скрывается за многими мифами и романтическими преданиями, которые время от времени будоражили умы многих пилотов: города-призраки, параллельные миры, путешествия во времени. Ходили даже легенды о том, что существует планета, о которой на древней Земле знали, как о Рае. "Темное Колесо" для современных звездолетчиков было такой же легендой, какой миф о Короле Артуре был для первых космонавтов.

- Но этого не может быть, - вздохнул Алекс, - он бы рассказал нам...

- Черта-с-два рассказал бы он. Его за то и убили, что он слишком много знал и он не стал бы подвергать опасности своих близких. Тот корабль не был пиратским. Просто Джейсон что-то нашел. И это что-то было настолько важным, что встревожило сильные круги.

- Что именно он нашел?

Рейф рассмеялся.

- Посмотри на меня, парень. Ты думаешь, что я это я? Нет, одна нога, кусочек печени и несколько мозговых клеток - вот все, что от меня осталось. Все остальное - это бионика и чудеса хирургии. Да, я тоже когда-то был Элитой, но сейчас мне чтобы плюнуть надо сосредотачиваться десять секунд.

Я больше не принадлежу его кругу. Он не сказал мне ничего, ведь я уже не вхожу в "Темное колесо". Но я не слепой и не глухой и делаю так, как мне говорят. И будь я проклят, если не сам Джейсон незадолго до смерти просил меня приглядеть за тобой, парень и подготовить к тому, чтобы ты смог пойти по его стопам.

Это было слишком велико для Алекса. Он молчаливо сидел у приборной панели, отрешенно перебирая пальцами приборы управления.

Наконец он собрался, улыбнулся и отбросил печаль и грусть.

- Хорошо, раз мой отец хотел этого, я не разочарую его.

(Продолжение следует)